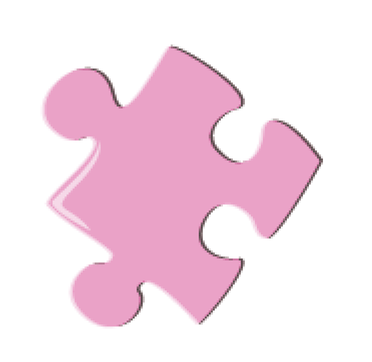


高等学校通识教育系列教材

高级数据库基础教程

叶小平 李强 陈瑛 叶晟 编著



清华大学出版社

高等学校通识教育系列教材

高级数据库基础教程

叶小平　李　强　陈　瑛　叶　晟　编著

清华大学出版社
北　京

内 容 简 介

本书是"高级数据库"课程的基础教材。全书突出计算机数据管理技术的逻辑主线，在编写思路和材料组织上具有体现整体架构、注重相互关联、彰示关键细节和强化实例讲解等特点。书中有选择性地介绍从经典数据库到当今的若干新型数据库的基本原理和相关技术，力求实现由经典关系数据库到新一代数据库技术的有效对接，并有助于完成由数据库理论技术的学习到从事相关研究探讨的基本过渡。

全书共10章，内容包括绪论、关系数据库基础、面向对象数据库和对象关系数据库、空间数据库与时态数据库、XML数据库、移动对象数据库、大数据技术简述、时态数据索引技术。

本书可作为高等院校计算机科学与技术及其相关专业本科生选修课和研究生必修课教材，同时由于主线突出和内容简明，也适合于相关人员作为自学材料参考使用。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

高级数据库基础教程/叶小平等编著. —北京：清华大学出版社，2018
(高等学校通识教育系列教材)
ISBN 978-7-302-50165-7

Ⅰ. ①高…　Ⅱ. ①叶…　Ⅲ. ①数据库系统－高等学校－教材　Ⅳ. ①TP311.13

中国版本图书馆CIP数据核字(2018)第112402号

责任编辑：刘向威　王冰飞
封面设计：文　静
责任校对：焦丽丽
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市铭诚印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　**印　张**：22.5　**字　　数**：546千字
版　　次：2018年10月第1版　**印　　次**：2018年10月第1次印刷
印　　数：1～1500
定　　价：59.00元

产品编号：069648-01

前　言

计算机处理的对象是数据，数据可以是以数字符号为基础的数值型数据和由字符、音频、视频乃至网页等组成的非数值型数据。作为一个学科，计算机科学技术主要应用于非数值型数据。在众多的非数值型数据的应用中，有一类称为数据密集型的计算机应用领域，其基本特征是涉及数据体量巨大、数据计算结果需要长久驻留计算机及数据保持大范围共享等。这是迄今为止最大的计算机应用领域，因为任何规模化的计算机信息管理系统都需要以其为底层技术支撑，这就是数据库技术。

计算机应用从技术实现角度来看，可以分为数据计算和数据管理。数据库技术属于数据管理的技术范畴。自 20 世纪 50 年代中期开始至今，数据管理经历了人工管理（基于应用程序）、文件系统管理（基于操作系统）和数据库管理（基于 DBMS）三段历史进程；而自 20 世纪 60 年代中后期开始至今，数据库自身也经历了由第一代数据库（层次和网状数据库）到第二代数据库（关系数据库）再到第三代数据库（以对象数据库为典型代表的各类新型数据库）的 3 个发展阶段。如果将关系数据库等看作是经典数据库，则通常可以将包括对象数据库在内及其之后出现的各类数据库通称为高级数据库或现代数据库。这些数据库或者基于数据模型的创新，或者出于应用维度的扩展，或者与计算机各类新鲜的主流技术密切结合，它们共同构成当今兴旺发达的整个数据库家族。这个数据库家族相当庞大，以致需要从各个不同角度进行审视和不同层面展开讨论才有可能理解与把握。本书主要是从数据模型及建立在其上的数据操作原理视角进行内容组织和展开叙述的。

本书共分为 10 章。第 1 章是绪论，简述数据管理的产生背景、技术路线，以及数据库技术在整个计算机科学技术领域中的地位和意义。第 2 章是关系数据库基础，由于高级数据库中许多概念、原理和技术都与关系数据库有着密切关联，深刻地认识和把握关系数据库技术对于其他数据库技术的学习与研究是不可或缺的。第 3 章和第 4 章讨论对象数据库技术，其中第 3 章的面向对象数据库是基于全新数据模型的数据库，可以看作是 C++ 基于数据库机制的扩充；第 4 章的对象关系数据库的数据模型基础仍然是关系模型，可以看作是 SQL 关于面向对象原理方法的扩充。第 5 章和第 6 章分别是空间数据库和时态数据库，可以看作是由于数据库应用领域扩大和应用层面深化而驱动数据库在空间和时间应用维度方面的扩展，这种扩展可能被局限于关系数据模型（如时态关系数据模型），也可能需要建立新的数据模型（如空间数据的镶嵌数据模型和矢量数据模型）。第 7 章是 XML 数据库，这是一种半结构化数据的管理技术，有别于结构化的关系数据与对象数据，通常需要建立反映出“数据与结构融合”自身特点的更为复杂的数据模型。第 8 章是移动对象数据库，这是为了适应由于网络技术发展和移动通信终端设备普及而带来的新的数据管理需求。第 9 章是大数据技术简述，从数据管理角度考虑，可以看作是一种范围更为广阔和内容更为新颖的分布

式数据管理技术，其中 NoSQL 实际上是将常规的数据库技术推向了一个新的阶段。第 10 章是时态数据索引技术。各类新型数据库通常都没有成熟 DBMS 支持，数据存取的有效途径多是基于研究和开发相应的数据索引，因此，数据索引也就成为高级数据库技术的基本内容之一。本章应用时态数据库相关知识建立了时态数据索引框架，并将其应用到 XML 数据索引和移动对象数据索引，这实际上可看作是本书主要内容的一个综合应用。

本书的第 1 章、第 3～7 章和第 10 章主要由叶小平编写，第 2 章主要由李强编写，第 8 章主要由叶晟编写，第 9 章主要由陈瑛编写，同时，陈瑛参与了第 5 章和第 10 章的编写，李强参与了第 9 章的编写，叶晟参与了第 7 章的编写。全书由叶小平负责统筹。在本书编写过程中得到汤庸教授的热情鼓励和大力支持，其中不少观点的提出和材料的选择都得到了汤庸教授的启示和帮助，在此谨致以衷心感谢！同时，书中参考和借鉴了较多的数据库方面相关专著、经典教材和科研论文，书中每章之后附有其中的主要参考文献，然而难以一一列举，作者对此表示歉意。由于本书涉及的许多内容已经成为经典，不少专著和教材中对其都有专项论述，加之本书性质定位所限，因而大多没有列举原始的文献资料，这里谨对本书涉及的相关书目和文献的专家学者们表示诚挚的谢意！

此外，林衍崇和陈钊滢等研究生也参与了本书部分内容的讨论和完成。

由于高级数据库领域范畴广而深邃，相关技术日新月异，即使本书所论及部分也难免挂一漏万，失之偏颇。加上编著者学识和经验所限，疏漏与不当之处在所难免，恭待专家学者和教学同行批评指正。

2018 年 5 月于

中山大学海滨红楼及广东东软学院 Lisp 园

目　录

第1章　绪　论

数据库可以看作是基于数据管理的计算机技术系统，一般而言，它由一组相互关联的数据集合和一组用于访问及操纵数据的计算机程序组成。数据库技术是计算机学科中发展最快和应用最广的重要领域之一。由于任何信息管理系统都需要有数据库的后台支持，因此数据库应用现在已经成为人们社会经济活动中不可或缺的核心技术支撑。同时数据库还通过互联网融入人们日常生活的方方面面，如ATM、网上购物、浏览信息和社交网络等。正是由于数据库技术在各类计算机应用中占有很大比重，专注数据库技术研制开发的Oracle早已成为当今最大的计算机软件公司之一，而其他具有重大影响力的计算机巨头，如微软和IBM也都以相应的数据库管理系统为其主打的支柱产品。本章简要介绍数据库技术的发展及其在整个计算机学科领域中的地位和意义。

1.1　数据及其特性

在计算机信息时代，“数据”是广泛使用的一个术语，但越是使用广泛的通常也是越难以明确定义的，因为它可能是所有相关概念的“源头”，或者说是元概念，抑或根本就无法进行严格定义和准确描述。“不幸”的是，“数据”与“信息”一样，就是这样的元概念。没有明确定义而又应用极其广泛，这一有趣现象事实上是普遍存在的，如“生命”“人”“智慧”“能量”和“质量”等都是如此。这类“伞形”概念实际上需要从其最常见、最有用特征和与其他相关概念最基本联系等方面进行适当描述和把握，从而以其为基础建立起庞大适用的体系。对于“数据”而言，人们只能如此处理。

1.1.1　数据概念

数据(data)一词来自拉丁文“to give”，意为“给”或“供给”。由此引申，数据可以看作是确定的事实，并且能从中推断出新的事实。

为什么会有数据？人之所以能够从一般灵长类动物中脱颖而出，就是因为逐步进化出能够描述、认识和利用客观事物和现象的基本能力。随着人类文明的不断进步，人们意识到仅仅使用一般的语言文字和图形图像描述他们所处的这个世界是不够精确的，这种描述对于发展科学技术乃至推动人类社会不断前行更是远远不够的。为了准确描述客观世界(如科学技术所必需的各种测量等)，也为了有效地展开社会经济活动(如货币使用和贸易交换等)，更为了充分改造和利用自然(如按照科学规律设计、建造机器和建筑等)，人们还需要“数据”这种特定的信息表述形式，并进行彼此间的交互。从本质上来看，人类的一切生产、交换等社会活动都可以说是以“数据”为基础而有效展开的，数据的出现和使用，应当是人类

文明的重大进步之一。

鉴于“数据”概念的重要性和基础性，通常需要从下述不同的角度来理解和掌握。

1. 从数据表现形式上考虑

从本源上考虑，数据是客观事物某种特征在人们意识中的反映，因此具有特定的表示形式。

(1) 广义数据：描述客观实体特征的各种实体或符号记录。例如，远古人类的小棍计数、结绳刻痕记事等以具体实物形式表示的数据；文明社会中以语言文字、声音图形和各类数字等具有不同抽象层级的符号形式对事物特征或数量上进行的描述等。

(2) 狭义数据：能够通过数字化编码进入计算机并由计算机进行处理的抽象符号集合。在当今的信息时代，人们通常从这种狭义角度理解和界定“数据”概念。

2. 从数据基本来源上考虑

按照数据的来源区分，数据可以有下述几种形式。

(1) 测量型数据：如上所述，数据首先源于人们认识和改造客观世界所必需的“直接”测量。作为“有根据的数字”，数据指的就是对客观世界测量结果的表述。测量是人类进行各种活动中不可或缺的基本手段，更是科学技术的必备基础。没有测量，就不会有数据；离开了数据，任何科学技术都会成为无本之木和无源之水。

(2) 计算型数据：数据可以作为测量结果直接使用，还可以将已有数据通过数据处理后得到新的数据，这是数据本身含义的体现，也是人们使用数据进程中的一个重大进步，因为有些数据根本不能通过直接测量获得，而只能通过对已有数据进行计算处理后而得到，如到太阳的距离(约 1.5 亿千米)和太阳内部的温度(2000 万摄氏度)等。这样就有了“原始数据”和“非原始数据”之分。

(3) 记录型数据：测量只是涉及客观世界中的事物，是数据最早的来源。随着人类科学文化技术的向前发展，记录极大地扩展了人类社会活动的深度和广度，为了丰富社会文化生活和保障文明传承，需要通过文字、图形图像、音频、视频和多媒体等记录人们自身的各类活动。在信息时代，这些记录大多需要借助于计算机系统进行存储、处理和管理，都需要转化为计算机意义下的数据。这样，数据就有了“测量”“计算处理”之外的第三个来源：由文本文字、图形图像、音频视频和多媒体等组成的“记录”数据。

3. 从数据、信息和知识关系上考虑

数据与信息一样都是元概念，难以进行严格逻辑意义上的定义。但从计算机应用角度来说，可对“数据”“信息”和“知识”三者关系进行描述，这种描述有益于对数据概念的理解和把握，在实际应用中也是行之有效的。

(1) 数据：通常可以描述为事实或观察的结果，作为对客观事物或其特征的某种形式上的归纳，主要用作未经加工的原始素材。数据的一个基本特征是在使用时需置于具体场景之中表明其语义。例如，“37”这个数据并没有表示任何意义(即语义)。但将它置于人体温度语境中，就表明了一个人的体温是 37℃；而将其置于人的年岁语境中，就说明一个人的年龄是 37 岁等。也就是说，数据需要解释语义，不能解释或没有语义的数据就没有使用价值。

(2) 信息：通常可以看作是具有明确语义的数据或数据整合体，信息会“明确”告知人们一定的含义，但不能保证该含义是否合适与正确。

（3）知识：通常可看作是经过人类的归纳、整理和加工，最终呈现某种规律的正确性信息。

数据、信息和知识在递进的链条上可以看作：在内涵上一个比一个明确有力，在表现上一个比一个丰富多彩，但归根结底，数据是这一切的基础。

4. 计算机程序和数据

经过多年的探讨和实践，人们认识到计算机科学与技术的主体是其中的软件原理研究、方法设计与技术开发。对于计算机软件而言，程序和数据是两个最重要的组成部分。因此，从某种考量出发可以认为，计算机软件正是由于其中的程序和数据才得以构成了真正意义上的计算机运行实体。

实际上，对于计算机软件来说，程序和数据通常是相互关联与密切整合的，但在实际应用中却有孰重孰轻和谁主谁次的考虑。为了讨论此项问题，需要先从不同角度对计算机数据进行适当的分类。

（1）数值型数据和非数值型数据。如前所述的整数、实数等基于测量和计算的数据就是数值型数据，其特点是可以通过转化为二进制数而“直接”进入到计算机并为计算机程序所处理。主要用于记录的字符、图形图像、声频视频及多媒体等数据都是非数值型数据，其特点是需要经过适当的编码方可进入计算机并为应用程序所处理。如今，非数值型数据已经成为所有计算机数据的主体组成。

（2）挥发性数据和持久性数据。从是否长期驻留计算机来看，可以将数据分为挥发性（transient）数据和持久性（persistent）数据。显然，存在于内存中且当相应程序结束就被“析构”的数据是挥发性的，而相应程序结束后会被“自动”建构存储在外存中的数据就是持久性的。

（3）私有性数据和共享性数据。从是否为多个程序共享同用来看，可以将数据划分为私有性（private）数据和共享性（share）数据。只能在个别特定程序中使用和处理的数据是私有性数据，能够被多个不同程序共同使用的数据则是共享性数据。显然，使用同一数据的应用程序越多，相应数据的共享程度就越高。

① 在直接使用程序设计语言解决实际问题的计算机应用过程中，程序是主体，数据是从属于特定应用程序的，此时的数据多是数值型数据，通常具有挥发性和私有性。

② 在各类涉及信息存储和管理的软件系统中，数据是主体，程序是围绕和服从于相关数据的，此时的数据大多是非数值型数据，通常具有持久性和共享性。

实际上，根据软件系统中程序是主体还是数据是主体，可以认为各类众多的计算机应用由“数据处理”和“数据管理”两大部分组成。

1.1.2 数据处理和数据管理

计算机的英文为“computer”，其原始含义是“计算工匠”。在最初时期，计算机应用的对象是“数”，此时“数据”就是“以数字形式表现出来的客观事物的特征证据”。这很自然，因为任何数字都可以“直接”转换为二进制数字，而数字计算机就是基于二进制数字的存储处理装置。此时如同工具是手的延伸一样，“computer”是人类大脑“计数”智力的延伸。ASCII码标准出现是数字处理技术中的划时代事件，它使得起源于数字“运算”的计算机技术能够应用到字符文本的处理。从此，就有了“数值型”数据和“非数值型”数据的技术之分。人类

思维需要借助语言来实现,而字符就是语言的载体,计算机应用进入由文字字符为代表的非数值型数据领域,为计算机具有真正意义上的"人脑智能"提供了可能,打开了计算机实现真正意义上的人脑"延伸"通道,此时,计算机才可以名副其实地称为"电脑"。

人类大脑的功能实际可以分为两个方面:一个是智慧,即处理问题的能力;另一个是记忆,即传承知识的能力。从数据角度考虑,计算机作为"电脑",其智力也突出表现在数据处理(即数据计算)能力和数据管理(即数据存储(记忆数据)检索(记忆数据的使用))能力上。

1. 数据处理

数据处理的操作通常可以看作是通过对已有数据进行"计算"或"运算"以获取新的有用数据。这些运算可以是加减乘除等算术运算和"或""与"等逻辑运算,也可以是更为复杂的计算机意义上的算法运算,如排序、查找和索引等。这方面内容集中体现在"数据结构与算法"课程当中,同时也普遍分布在计算机的各个领域与技术实现当中。数据处理计算具有下述特点。

(1) 算法复杂性。算法内容复杂深入,算法设计灵活多变,但计算涉及的应用范围都有相对窄小的边界。

(2) 基于程序设计语言。通常都需要借助某一种高级程序设计语言实现相应的数据处理。

(3) 数据量相对较小。计算数据多是基于键盘输入,因此计算过程中涉及数据量相对较小。

(4) 数据的挥发性和私有性。数据没有长时间存留和大范围多程序共享的一般需求。

2. 数据管理

数据处理计算是计算机最重要的应用之一,可以看作是一种"CPU 密集型"(CPU intensive)应用,另一类更为广阔的被称为"数据密集型"(data intensive)的应用领域就是数据管理。数据管理着眼于数据的持久存储、高效查询和大范围共享互用等,因此具有下述突出特征。

(1) 数据量巨大。巨大的数据量需要存储在外存储器当中,在计算机运行过程中内存只能装载其中很小的一部分数据。

(2) 数据持久性。与数据计算处理不同,管理过程中涉及的数据需要长期驻留计算机系统。

(3) 数据共享性。系统管理的数据为众多应用程序或应用单位等大范围共享。

数据管理具体涉及数据收集整理、组织存储、维护传送和查询检索等数据操作,包括管理信息系统、办公室自动化系统、人事管理系统、酒店预订管理系统和金融信息系统等方面,实际上已经形成迄今为止最大的计算机应用系统。自从计算机由主要从事数值型数据的科学计算转变到从事更为广泛的非数值型数据应用以来,数据管理就已在计算机科学技术领域占据重要的核心地位。

1.1.3 数据管理和数据库

现在,整个计算机科学技术实际上几乎都以非数值型数据为基本应用对象,而其中非数值型数据管理已经成为最大的一类计算机应用领域。当一个计算机软件系统具有了数据共

享、数据独立乃至最重要的数据模型时,就可以看作是具有数据管理系统的基本特征。由此通常认为,自计算机科学技术诞生发展以来,数据管理技术经历了人工管理(应用程序管理)、文件系统管理(操作系统管理)和数据库管理(专用 DBMS 管理)三段历史进程。

1. 人工管理

人工管理实际上就是人们通过编写应用程序进行数据管理,其基本特点是一组数据对应一个特定应用程序,当多个不同程序使用同一数据集时,需分别设计数据结构,无法自动关联和相互参照,需要人工进行干预处置,因此也称为基于程序的数据管理。由此会导致下述问题。

(1) 数据共享性不足。同一数据在不同程序中需要各自设计逻辑与物理结构及相应存取方式,难以进行有效的数据共享。

(2) 管理工作重复进行。数据使用过程中出现大量冗余,从而导致需要对冗余数据进行重复管理,共享性品质较差。

(3) 数据独立性差。数据的逻辑结构和物理结构交叠影响,使得数据与程序关系密切,当数据本身发生改变时,相应管理程序必须改变,数据缺乏基本的独立性。

基于程序的数据管理主要出现在 20 世纪 50 年代中期之前,当时没有磁盘等可直接存取的必要设备和操作系统支持等技术条件,因此应用程序(即人工方式)也只能是当时对数据进行管理的唯一可行办法。

人工管理方式如图 1-1 所示。

2. 文件系统管理

基于文件的数据管理主要出现在 20 世纪 50 年代末期到 20 世纪 60 年代中期,实际上就是使用操作系统中专门的文件系统完成相关工作。文件系统管理具有下述特点。

(1) 数据长期驻留。计算机磁盘和磁鼓等提供了长期保存数据的硬件条件,文件系统提供了数据在外部存储器多次进行查询和更新的软件环境。

(2) 一定程度数据独立性。应用程序与数据之间由文件系统提供的存取方法进行转换,数据与程序之间具有一定独立性。

(3) 一定程度的数据共享性。数据按照内容、结构和用途组成文件,而文件面向应用,可以为一组使用同一数据的应用程序所共同使用。但当不同应用只具有部分相同数据时则需要建立不同的数据文件,此时又回到了程序管理的情形。

基于文件系统的数据管理如图 1-2 所示。

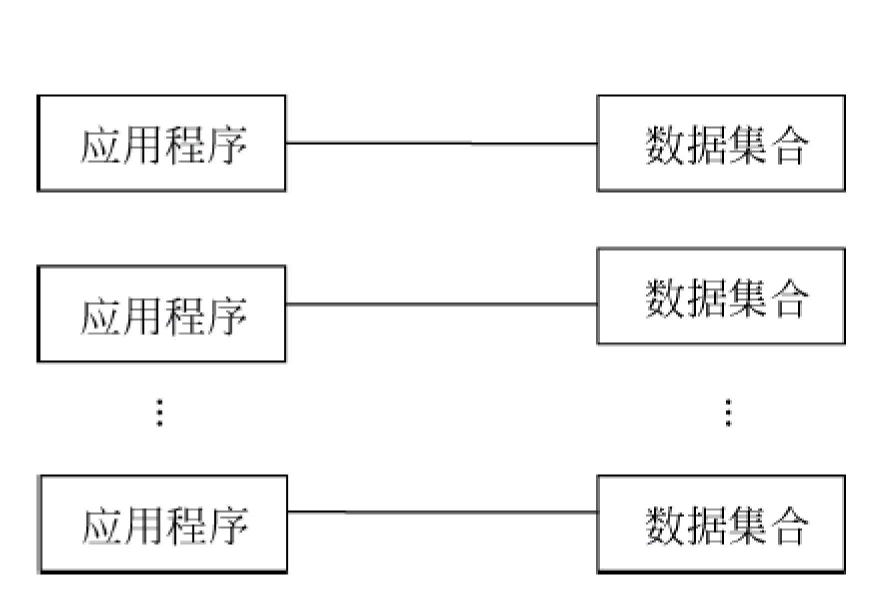


图 1-1　应用程序(人工)数据管理

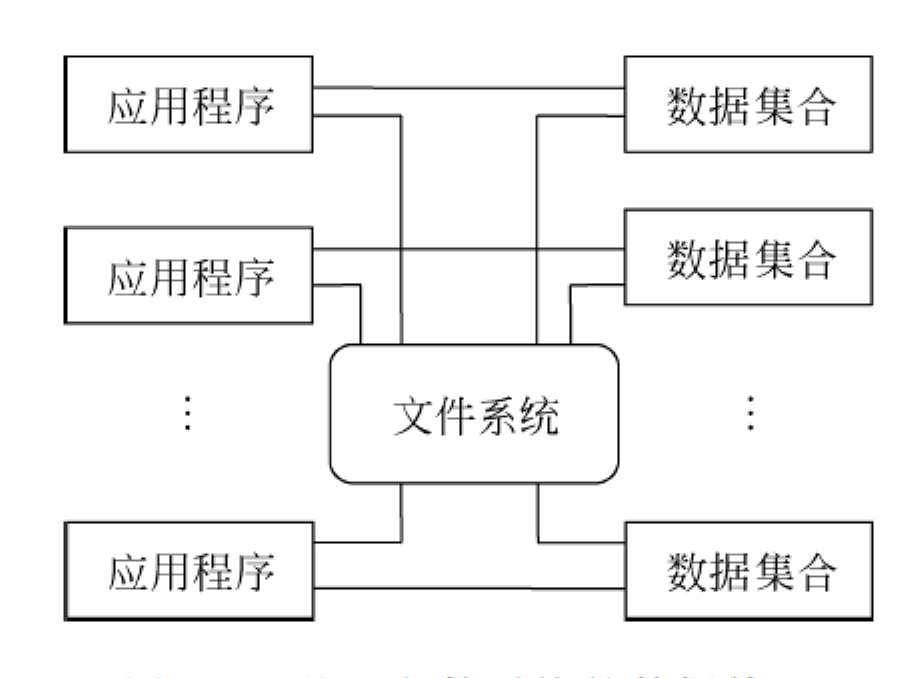


图 1-2　基于文件系统的数据管理

操作系统以专门的文件系统软件对数据进行操作，提供较人工阶段更为有效的数据管理模式。由于只具有部分的数据独立性，文件系统中数据冗余仍然较大，数据共享性也不够理想。随着对数据管理性能要求的提高，如更高的共享性、更好的独立性和更有效的数据查询与数据更新等实际需求，推动着数据管理的方法和技术朝新的方向不断提升和突破，数据库技术应运而生。

3. 数据库管理

通过应用程序和文件系统进行数据管理的实践进程，人们逐步认识到数据的有效管理实际上就是数据的结构化管理，这是因为在计算机系统内，数据和文件的简单堆积将缺乏使用价值。具体而言，由于实际应用中的数据结构复杂、数据量巨大，简单依靠应用程序乃至操作系统中的文件系统对数据进行管理使用存在着很大缺陷，需要有建立在操作系统之上的专门软件系统，这就是以统一管理和共享数据为设计目标的数据库管理系统（Database Management System，DBMS）。

1）数据库数据管理特征

数据库系统出现于 20 世纪 60 年代末直至现在仍在使用，它具有下述基本特征。

（1）数据共享性。数据作为整体应用单位的共享资源由 DBMS 统一管理。这种管理不依赖任何个别应用程序和个别用户，能够在系统级别上确实保证和真正实现数据的通用共享。

（2）数据独立性。数据由 DBMS 统一调配使用，用户与数据管理在真正的逻辑和技术层面上实现了数据独立。由于其独立性导致数据存储和组织等细节透明，从而使得用户可以在更高的抽象层面上审视和访问数据库中的存储数据，为共享性提供了必需的技术支撑。

（3）数据规范性。统一管理数据之后，系统能够立足于全局结构更加合理地组织和更为有效地调配数据，能够最小程度地减少数据冗余，更合理地设计和实现数据的标准化与规范性，从而更加有利于数据的转移传输和更大范围内的共用共享。

（4）管理完备性。由于面对整个应用单位而非个别用户，因此 DBMS 能够研制得更为复杂庞大，从而具有更加多样和更为有效的功能。事实上，现有 DBMS 功能已经不仅仅限于一般的数据存储和查询，还具有查询优化、数据库保护（完整性与安全性）和事务管理（并发控制和故障恢复）等一整套完备机制，DBMS 已经成为在层级和规模上都不逊于操作系统（OS）和办公自动化系统（OA）的大型系统软件。

数据库系统管理数据情形如图 1-3 所示。

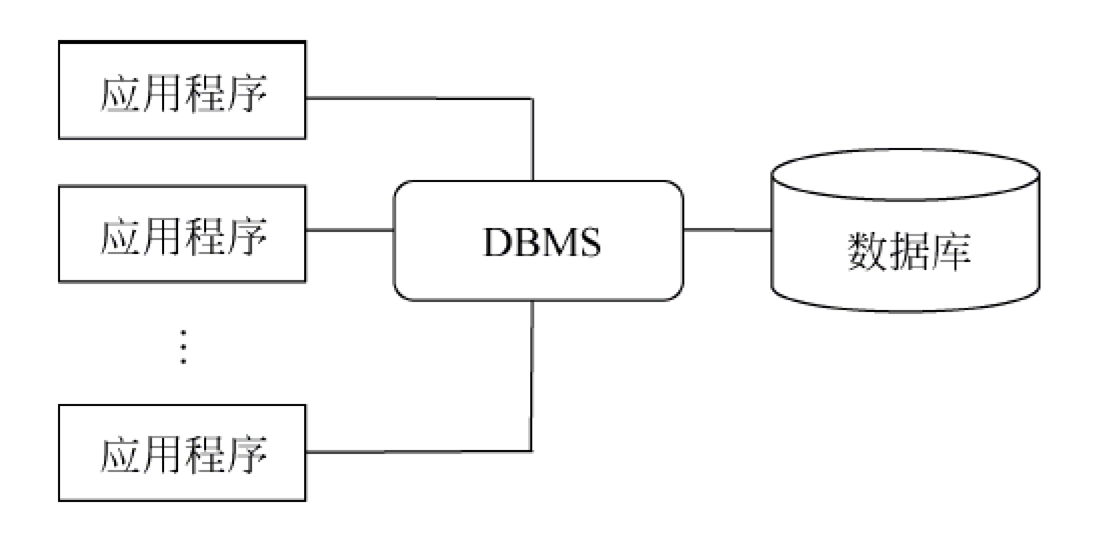


图 1-3　基于数据库的数据管理

数据库技术是计算机学科中发展最快的应用领域之一，也是应用广泛的计算机关键技术之一。自 20 世纪 60 年代中期至今，四十多年的发展经历了三代演变过程，如今已经成为

以数据建模为核心概念、以数据库管理系统为关键技术和以数据库系统为各类信息管理系统后台技术支撑的内容精深和应用广泛的计算机学科领域，带动了 DBMS 这样一个巨大的计算机软件产业及其相关产品。人们称数据库系统在整个计算机发展进程中地位突出也许不能算作是言过其实。

2）数据库管理系统

如上所述，数据库数据管理的基本特征是数据的集成统一管理和数据的有效共享同用，需要有一个大型的系统软件来实现其基本功能，这就是数据库管理系统（DBMS）。由专门的软件系统而不是借助于操作系统中内嵌的文件处理功能管理数据是数据库系统与其他任何软件系统区别的根本性标志，一般数据管理过程中的共享性、独立性等就有了真正技术层面的基本保障。

（1）数据独立性实现。在 DBMS 管理之下，众多用户的数据库应用程序不必介入数据文件的打开、关闭、读/写和存储等相对低级操作，也就是说可以脱离数据存储读/写及其技术细节实现，也不用担心被 DBMS 所屏蔽的物理层面上文件结构的改动，可以在更高的抽象层级上和更广泛的语义范畴内组织观察数据并提出自身操作需求。

（2）数据共享性实现。通过 DBMS 对数据的统一管理，用户可以从全局的理念出发合理组织数据，减少数据冗余，在真正意义上实现高品质的数据共享，这是由于如果不能对数据实施整体统一的观察和组织，数据就会出现交叠或重复，经过数据更新之后就会出现数据不一致的情形，不同用户得到的数据信息就可能出现差异从而导致共享数据丧失应有的作用和意义。此问题只能在 DBMS 框架内才能得到有效解决，如关系数据库中模式设计课题。

（3）基本保障功能实现。数据管理最根本的应用目的就是实现数据共享也就是多用户数据查询功能。在实际应用和维护过程中，还需要其他一些基本机制予以保障。例如，模式设计机制以保障数据共享本意的实现，共享性带来的安全性机制以保障共享性带来的数据享用权限，故障检测恢复机制以保障共享性带来的系统故障多发性，并发机制以解决多用户并发访问引发的语义和技术上的各类问题处理等。由于 DBMS 不是只为个别应用程序服务，作为系统软件自然就可以综合设计整体研发出能够实现围绕数据共享与数据独立的各类相关机制，而这在文件系统中是难以企及甚至是根本无法实现的。由于面向整体，如同操作系统那样，具体技术细节向用户透明，对于 DBMS 来说，即使相应软件系统做得更大和更复杂，用户也都是可以接受的，而实际情况也确实如此。

DBMS 是基于数据库管理数据的核心部件，其对数据库管理数据机制和功能的有效实现有着决定性作用。通常一种 DBMS 都和一种数据库语言相互对应，而 DBMS 最重要的关键点就是合法、正确、安全和有效地执行相应的数据库语言。例如，关系数据库管理系统（RDBMS）就可以看作是 SQL 的一个平台实现，同时还可以按照不同用户需要提供诸如交互式 SQL、嵌入式 SQL 和动态 SQL 等多样化的用户接口。

1.2 数据库技术发展概述

数据共享性是数据管理的基本要求和应用驱动，数据独立性和数据集成统一管理也都有基于共享性的考量。如果缺乏独立性，所存储数据的逻辑与物理结构都将依赖于用户的

应用程序，而由于用户需求的多样性和易变性会使得系统变得极其复杂和不够稳定，从而就难以保障共享性的有效实现。此外，共享性自然需要存储保管尽可能多的相关数据，而这大量的数据不能简单堆积在相关存储器中，否则相关的使用（如查找）就非常困难，或者根本就不可能完成，由此就难以实现众多不同用户对存储数据的共享性需求，因此必须对所涉及数据进行集成化的统一管理。进行集成化统一管理的基本技术途径就是将数据按照相互间的内在逻辑关联组织起来，并通过适当机制将这种逻辑结构转化为在机器上实现的物理结构，也就是说，数据的集中管理本质上就是需要建立起数据集合上的数据结构。有了顶层框架层级上的数据结构就能够在其内部统一定义系统技术级别上的所有用户共用的数据操作，以及保障共享顺利实现的各类约束条件，这就是人们所熟知的由数据结构、数据操作和数据约束构成的数据管理模型或称数据模型。在一定意义上可以认为，以数据模型为支撑而实现的数据管理系统就是数据库系统。因此，人们通常都是从所依据的数据模型出发，对数据库技术的发展历程进行分段。按照此种观点，数据库技术的发展经历了第一代数据库（层次和网状数据库）、第二代数据库（关系数据库）和新一代（或第三代）数据库（以对象数据库为代表）等 3 个发展阶段。

1.2.1 格式化数据库

第一代数据库系统包括层次和网状数据库系统，它们分别基于层次数据模型和网状数据模型。由于层次数据模型对应于树形结构，网状数据模型对应于有向无环图结构，具有明确的格式化的“数学”描述，所以也统称为格式化数据模型，相应的数据库称为格式化数据库。

1. 层次数据库

层次数据库主要代表是美国 IBM 公司于 1969 年研制成功的世界上第一个商品化 DBMS 产品 IMS(Information Management System)。

层次数据库采用树形结构表示所涉及的实体型及相互关系（数据结构）。其中结点表示一个实体型（记录），通常每个结点都由多个数据项（字段）组成，如图 1-4 所示。

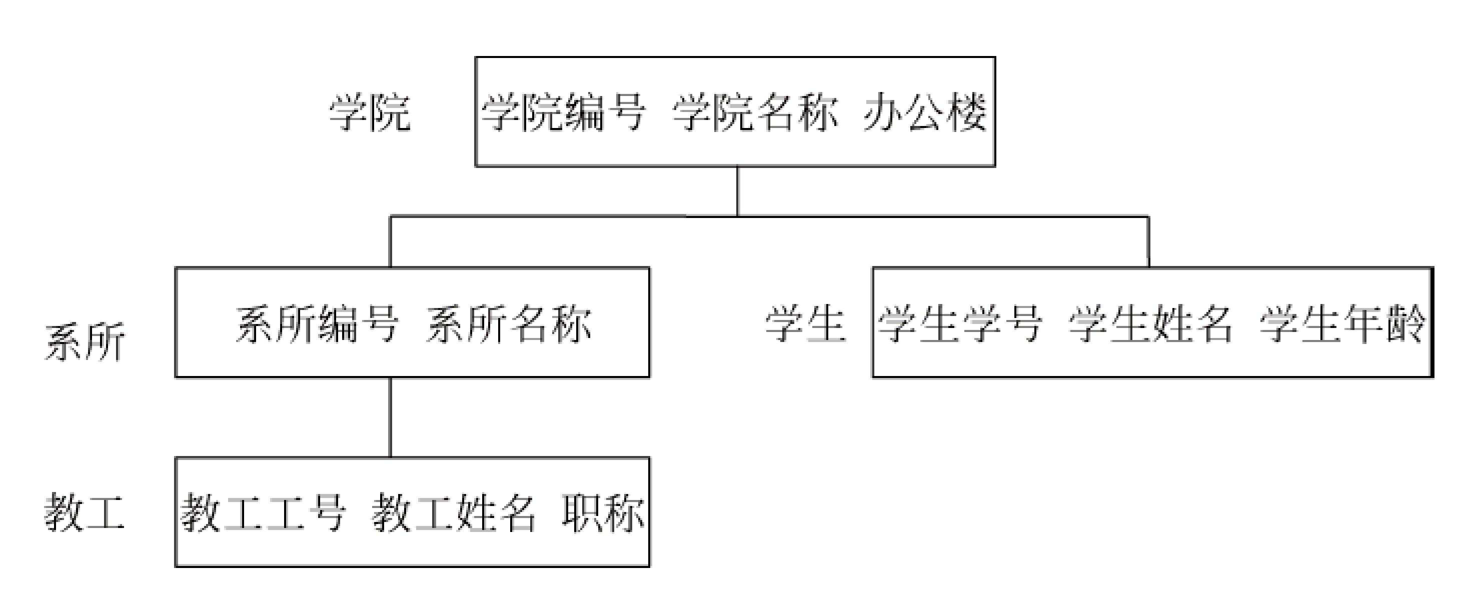


图 1-4 层次数据的树形结构

作为一种树形结构，层次数据库能够进行由根结点到叶结点的单向查询，因此适合于表示和访问“一对多”的数据关联；在处理“多对多”联系时则需要进行适当转换，并带来较多的数据冗余，同时在实现数据操作过程当中会受到较多的限制。

2. 网状数据库

美国 CODASYL(Conference on Data Systems Languages)协会下属 DBTG(Data base

Task Group)于1969—1970年对数据库技术进行了系统研讨，提出了DBTG报告。该报告首次确定了数据库系统中许多基本概念、方法和技术。由于其出发点是基于网状数据模型，通常将其看作网状数据库设计的代表之作，因此，网状数据模型也称为CODASYL模型或DBTG模型。网状数据库的原型主要有以下两类。

(1) 美国通用电气公司Bachman等于1964年开发的IDS(Integrated Data Store)系统，它奠定了网状数据库系统基础，也是世界上第一个成功实现的DBMS。

(2) 20世纪70年代中后期，典型网状数据库系统有Honeywell公司的IDSII，HP公司的IMAGE等。

网状数据库数据模型基于图结构，图中每个结点表示一个实体型记录，如图1-5所示。

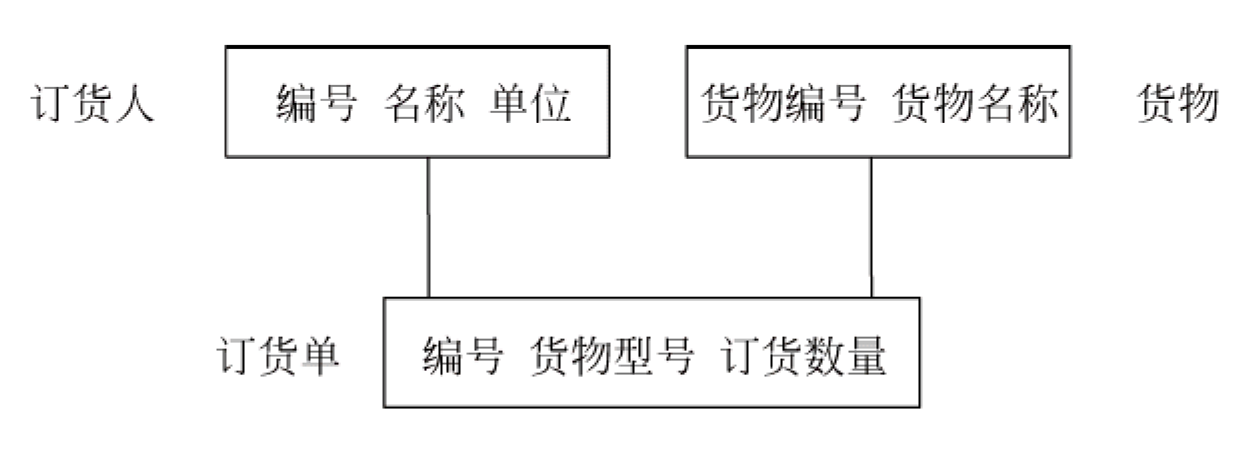


图1-5 网状数据的图结构

网状数据库能够更为直接地描述客观世界的真实情形，存取效能较高。作为图形结构，数据关联表示比较复杂，使用不够方便，同时数据独立性也不尽如人意。迄今为止某些网状数据库管理系统还在继续使用当中。

3. 格式化数据库意义

格式化数据库在数据库发展史上曾经发挥过重要作用，通常认为层次数据库是数据库系统的历史先驱，网状数据库则奠定了近现代数据库理论与技术基础。格式化数据库在数据库发展进程中具有下述基本意义。

(1) 支持数据库三级模式体系结构。数据库三级模式体系结构在整个数据库技术中的地位类似于冯·诺依曼体系结构在整个计算机体系结构中的地位。

(2) 通过存取路径表达数据联系。厘清了文件系统与数据库系统的根本界限，即数据库系统在存储数据本身的同时也存储数据关系。

(3) 建立独立于常规编程语言的数据定义语言。这种数据库语言用于描述和表示数据库的外模式、模式和内模式及其相互映射。

(4) 导航式数据操作语言。格式化数据库的数据查询与操纵语言具有一次一记录的过程化特征，可以嵌入到常用的高级程序设计语言当中。

1.2.2 关系数据库

关系数据库系统是数据库发展历史上的第二代数据库系统。

1. 第二代数据库

关系数据模型是由IBM公司San Jose研究室研究员E. F. Godd于1970年在其经典论文"大型共享数据库数据关系模型"中首次提出的，由此开辟了数据库关系技术和关系数据理论研究的基本方向，为关系数据库奠定了坚实的数学基础。1974年，数据库界开展了一场分别以E. F. Codd和C. W. Bachman为代表的支持与反对关系数据库大辩论。辩论的直

接后果促进了关系数据库的迅猛发展，吸引了更多公司和研究机构对关系数据库原型进行研究，研究成果成批出现。

(1) 1976 年，IBM 公司发布 System R (1974—1980)，美国加州大学伯克利分校发布 Ingres 关系数据库系统。在当时各类关系数据库原型中，这两个系统功能较强，技术上也更具代表性，它们为关系数据库提供了比较成熟的技术，为开发商品化关系数据库软件创造了有利条件。

(2) 1979 年，美国 Oracle 公司推出了用于 VAX 小型机上的关系数据库软件 Oracle (v2.0)，这被认为是第一次实现了使用 SQL 语言的商品化关系数据库软件。

(3) 1981 年，Ingres 公司推出了商品化的 Ingres 关系数据库。

(4) 1982 年，IBM 公司在 System R 的基础上推出了 SQL/DS，并在 1985 年又推出了 DB2。它们也是两个商品化关系数据库系统。

由于关系数据库原型和商业化系统大都在 20 世纪 70 年代后期相继推出，因此，通常将 20 世纪七八十年代称为数据库时代。实际上，到了 21 世纪的今天，人们所使用的数据库系统大部分仍是关系型的数据库系统，关系数据库系统迄今在数据库发展进程中依旧辉煌。

关系数据模式如表 1-1 所示。

表 1-1　关系数据模式——1NF

学号	姓名	年龄	住址	邮箱	电话

2. 关系数据库意义

关系数据库在数据库发展进程中具有里程碑意义。

(1) 奠定关系数据模型理论基础。使得数据库技术从此建立在严格的数学支撑之上，数据库学科从应用技术走向了科学技术。

(2) 开发数据库专用语言。基于关系运算(关系代数和关系演算)的关系数据库语言 SQL 改变了格式化数据语言的导航查询方式，以联想和非过程的简洁易学风格受到广大用户欢迎，为数据库语言标准化打下基础。

(3) 解决数据库实现过程中的关键技术。通过研制大量关系数据库管理系统原型，提出和解决了查询优化、事务管理(并发控制与故障恢复)及安全性等系统实现过程中的一系列关键机制，极大地丰富了数据库理论和技术，促进了数据库技术和产品的蓬勃发展和广泛使用。

1.2.3　新一代数据库系统

语法和语义是计算机理论和技术广泛涉及并需要着力处理的两个基本方面。一般说来，语法面向机器，语义面向应用。关系数据模型可以用于描述现实世界数据某些逻辑结构和相互关联，但随着计算机应用领域的扩大，其重于语法而疏于捕捉和表达数据实体具有的丰富而重要语义的局限也日益显现出来。自 20 世纪 80 年代末以来，出现了以对象数据库为特色的新一代(第三代)数据库系统。

1. 对象数据库

面向对象设计理念可以看作是基于语义模型，它在计算机的各个应用领域都产生了重要而深远的影响，也为当时不断遭遇挑战的数据库技术带来新的机会和希望。

自 20 世纪 80 年代开始，人们就开始了基于对象的数据库系统研究，有关数据模型和数据库系统的研发基本上沿着 3 条路径展开。

（1）面向对象数据库。以面向对象程序设计语言为基础，研究持久性的程序设计语言以实现对象支持，这就是面向对象数据库系统。

（2）对象关系数据库。以关系数据库和 SQL 为基础扩展关系模型以实现对象支持，这就是对象关系数据库系统。

（3）纯对象数据库。建立完全不依赖于现有技术方法的面向对象数据库系统，由于各种原因，迄今还没有系统原型实现。

表 1-2 和图 1-6 所示分别为对象关系数据模式和面向对象数据模式。

表 1-2 对象关系数据模式——非 1NF

学号	姓名	年龄	住址	联系方式		
				邮箱	电话	微信

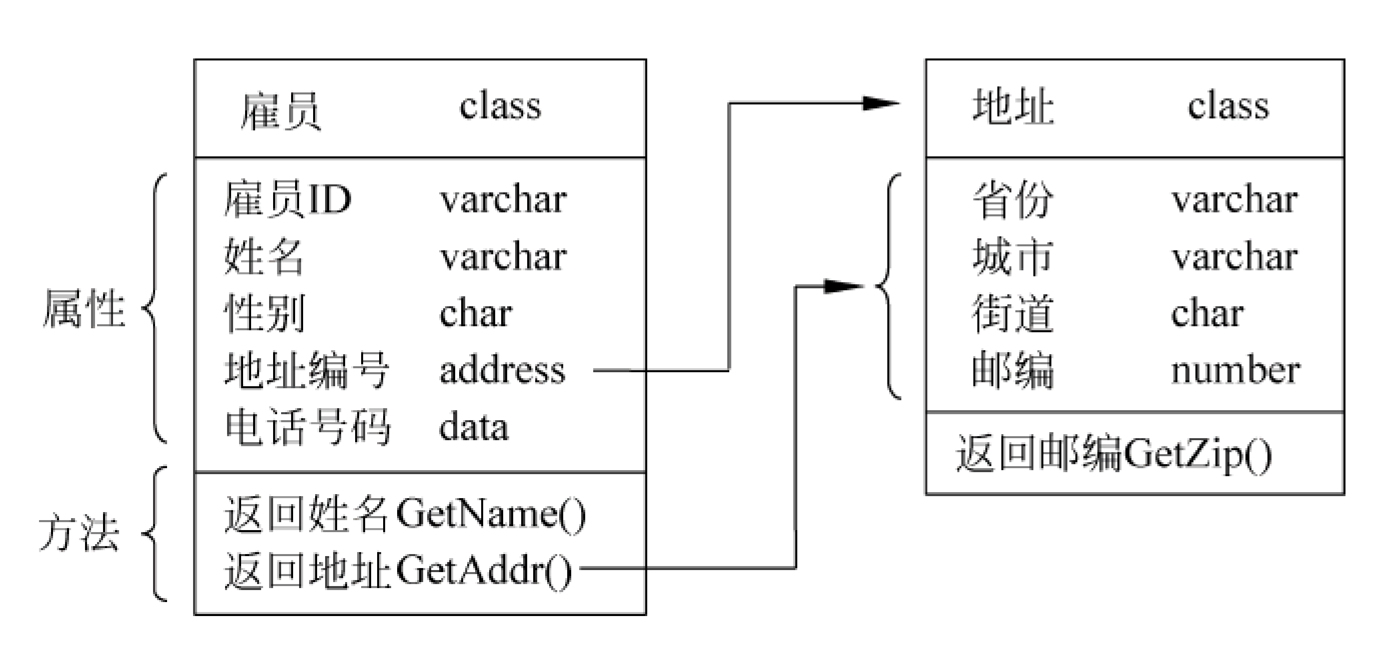


图 1-6 面向对象数据模式

2. 对象数据库特征

人们原先期望对象系统能够像关系系统取代格式化系统那样取代关系系统，从而使得对象数据库系统成为第三代数据库的“天下共主”，然而历史却没有能够重演。在这种情形下，人们对什么是第三代数据库系统实际上并未达成普遍共识，即还不存在一个公认的第三代数据模型。1990 年，高级 DBMS 功能委员会发表了《第三代数据库系统宣言》，提出第三代 DBMS 应具有如下基本特征。

（1）支持数据、对象和知识统一管理。第三代 DBMS 需要支持更加丰富的对象结构和规则管理，应当将数据管理、对象管理和知识管理整合为一体。

（2）保持或继承第二代数据库原有技术。第三代 DBMS 需要保持关系数据库现有技术（非过程化、数据独立性和事务管理等），不但具有良好的对象管理和规则管理，还能更好地支持原有管理与多数用户需要的即时查询等。

(3) 对其他系统保持开放。第三代 DBMS 需要对其他系统保持开放,即支持数据库语言标准、支持网络协议,具有良好移植性、可连接性、可扩展性和互操作性等。

尽管人们在第三代数据库上还没有达成基本共识,但对象系统的出现和发展却导致众多不同于前两代的重要系统诞生,从而构成了当今蓬勃兴旺的数据库大家族。这也是对象系统应有的历史功绩,也正是由于在第三代数据库定位上没有取得一致认识,业内通常也将有别于格式化和关系数据库的各类新型数据库系统都统称为新一代数据库系统。当今实在难以使用一种数据模型去统领广泛而深入的数据库应用领域,因为新一代数据库不像前两代数据库系统那样具有统一的标准数据模型。当前,根据不同应用需求,人们关注的数据模型主要有面向对象数据模型、对象关系数据模型、半结构化数据模型和 XML 数据模型及移动对象数据模型等。通过这些数据模型,结合各种计算机应用新技术,根据新的数据处理需求,出现了真正意义上的蓬勃旺盛、欣欣向荣的数据库大家族。基于树形模型的 XML 数据模式如图 1-7 所示。

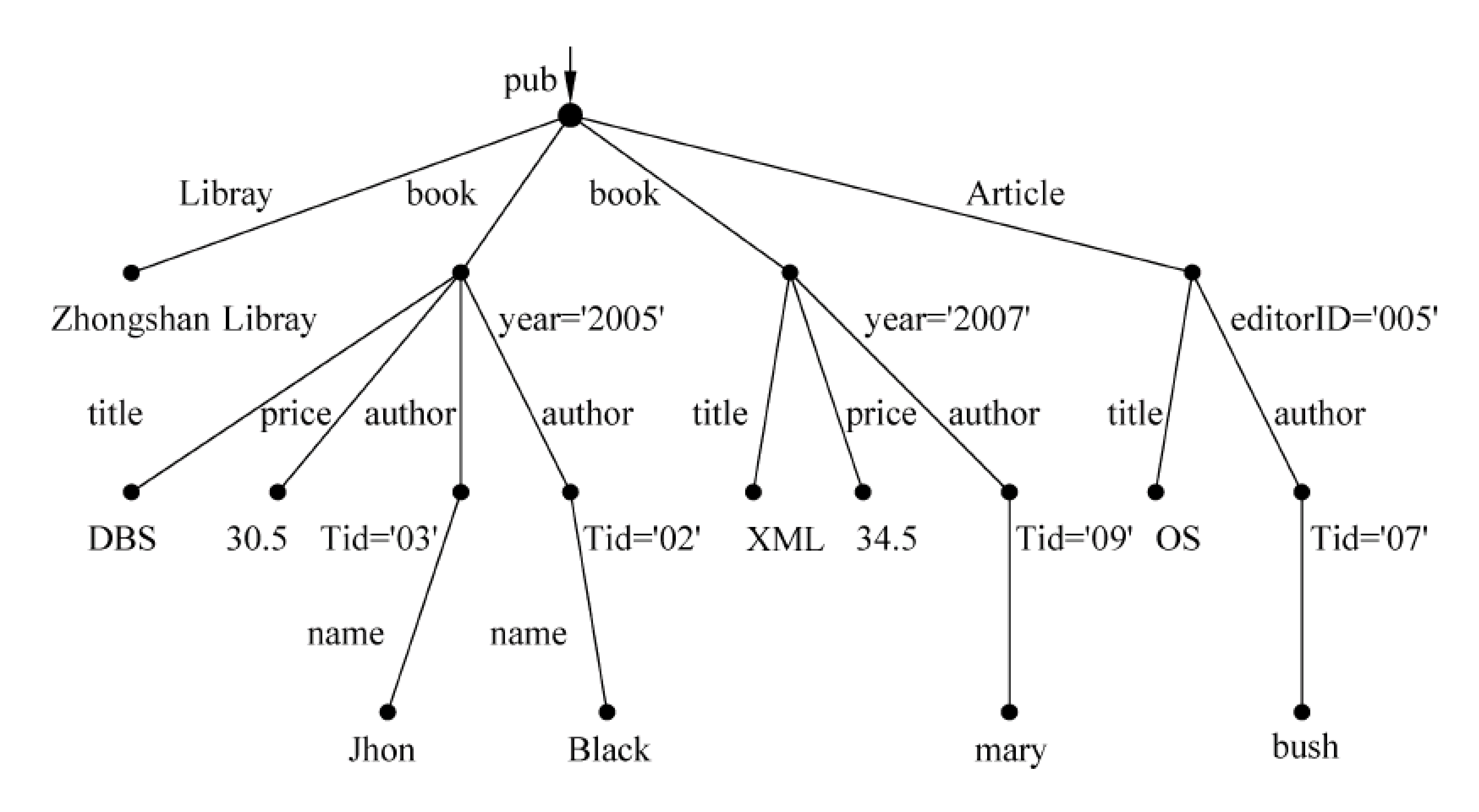


图 1-7　基于树形模型的 XML 数据模式

1.3　发展特征与驱动要素

数据库技术是几乎所有计算机应用技术发展和应用的基本保障之一,反过来,顺利解决计算机应用领域中提出的各种数据管理新的重大课题又会极大地深化扩展数据库应用领域,为数据库技术注入新的更强活力。

1.3.1　数据库技术发展特征

数据库技术发展特征主要表现在下述 3 个方面。

1. 数据模型研究深入

数据模型是数据库技术的核心与基础。例如,关系数据模型的提出既表现出数据库理论的巨大突破和飞跃,又体现出理论研究成果到实用化技术的出色转换,还带来了数据管理关键技术的有效攻克和商业化产品的巨大成功。面对广泛的数据管理需求和受到关系模型成功的鼓舞,随后人们又陆续研究和提出了各种新的数据模型,从而使数据模型成为整个数

据库新技术发展的最重要特征。

(1) 后关系数据模型。关系模型的关系范式 1NF 限制了关系数据库系统在新应用领域中的功能发挥,针对这种情形,通过引入适当的数据类型构造器,增强其结构建模能力,使得经过扩展后的模型能够表达比较复杂的数据类型。这里存在以下两种扩充思路。

① 基于结构扩充:这种扩充得到的扩展模型称为嵌套关系模型(NF2),可以表示"表中有表"的情形,并且表中一个域可以是一个函数(虚域)。

② 基于语义扩充:这种扩充既支持关系间的继承关系,也支持在关系上定义函数和运算,但关系结构仍是平面表,如美国加州大学伯克利分校的 Postgers 系统。

上述扩充中,数据表示的基础与核心仍然是"关系表",因此也称其为后关系模型。

(2) 面向对象数据模型。使用对象理念和方法来描述现实客观实体的逻辑组织、数据约束、相互联系等。面向对象的一系列核心概念与方法通过适当解释就构成了面向对象数据模型基础。面向对象数据模型具有丰富的语义表示和对象间多种相互联系,同时支持层次结构和多态重用。

(3) 对象关系数据模型。关系模型与对象基本原理与方法的适当整合,主要是在关系基础上扩充了对象模型的某些功能。基于对象关系模型的数据库系统既满足了传统应用的要求,又能适应大多数复杂数据处理,得到数据库大厂商支持。当前大型商用数据库系统大多具有对象关系的处理功能。SQL3 相对于以前 SQL 的最大扩展就是增加了面向对象数据类型,其中最重要的就是结构类型(行类型)和抽象数据类型。

(4) XML 数据模型。XML 数据是自描述和不规则的,可以通过适当的图模型进行表述。通常图中的边标记为元素的标识名,内部结点为属性-值对应集合,叶结点为元素文本内容。为减低表示复杂度,通常每个结点都有一个对象标识符。另外,XML 图模型需要有一个根结点。

(5) 半结构数据模型。半结构化数据存在着一定结构,只不过这种结构具有以下特点。

① 描述结构的元语言与描述数据的数据语言相同。

② 结构随时间动态变化。

③ 结构复杂不能纳入传统框架。

XML 数据属于半结构化数据范畴。当前半结构数据模型有基于逻辑和基于图的两种形式。

基于逻辑方式:通过描述逻辑和 Datalog 规则表示数据模型。

基于图方式:通过 OEM 模型表示,用于从半结构化网页中提取信息。

(6) 移动对象数据模型。从逻辑上可以看作是属于时空数据模型范畴,但由于自身特性,实际上已经形成了独特的研究方向与技术领域。

① 通常以移动点对象为主要研究数据对象,重点关注于移动对象的位置数据信息。

② 来自位置数据信息服务的应用驱动,数据模型不仅能够表示"过去"语义的数据,还需要描述"当前"和"未来"一段时间内的数据信息。

③ 由于在时间域上的"全方位"要求,以及对"实时性"和"预测性"的特殊需求,相应存储和管理"当前"和"未来"技术与传统方法有着明显区别,存在着重大创新。

2. 与计算机新技术融合

数据库技术发展的一个显著特征是数据库与计算机新技术紧密结合。随着计算机各种新技术相继涌现与不断发展,提出了各种各样新的需要处理的数据形式,由此形成了数据库

领域许多新鲜分支和研究课题,丰富和发展了各种新型数据库技术。数据模型与计算机应用技术结合情况如图 1-8 所示。

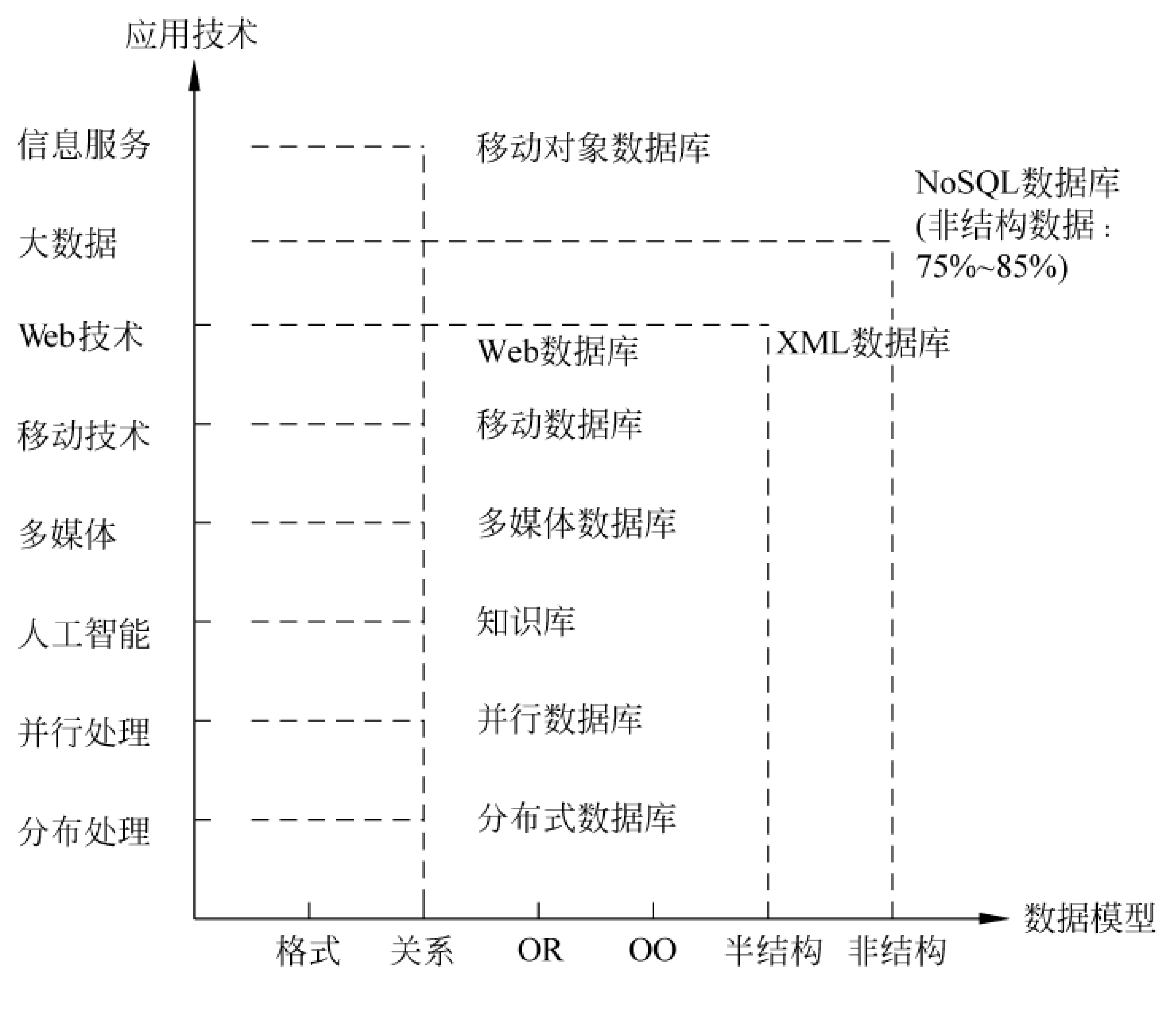


图 1-8 数据模型与计算机应用技术结合

3. 面向新型应用领域

与应对各类应用技术提出新的数据类型需求处理不同,许多特定的计算机应用领域,或者需要综合应用多种数据库技术,或者需要对大量本身已有数据集合进行新视角审视与处理,从而使得数据库技术应用范围不断扩大,新型数据库系统不断涌现。数据模型与数据处理新需求结合情况如图 1-9 所示。

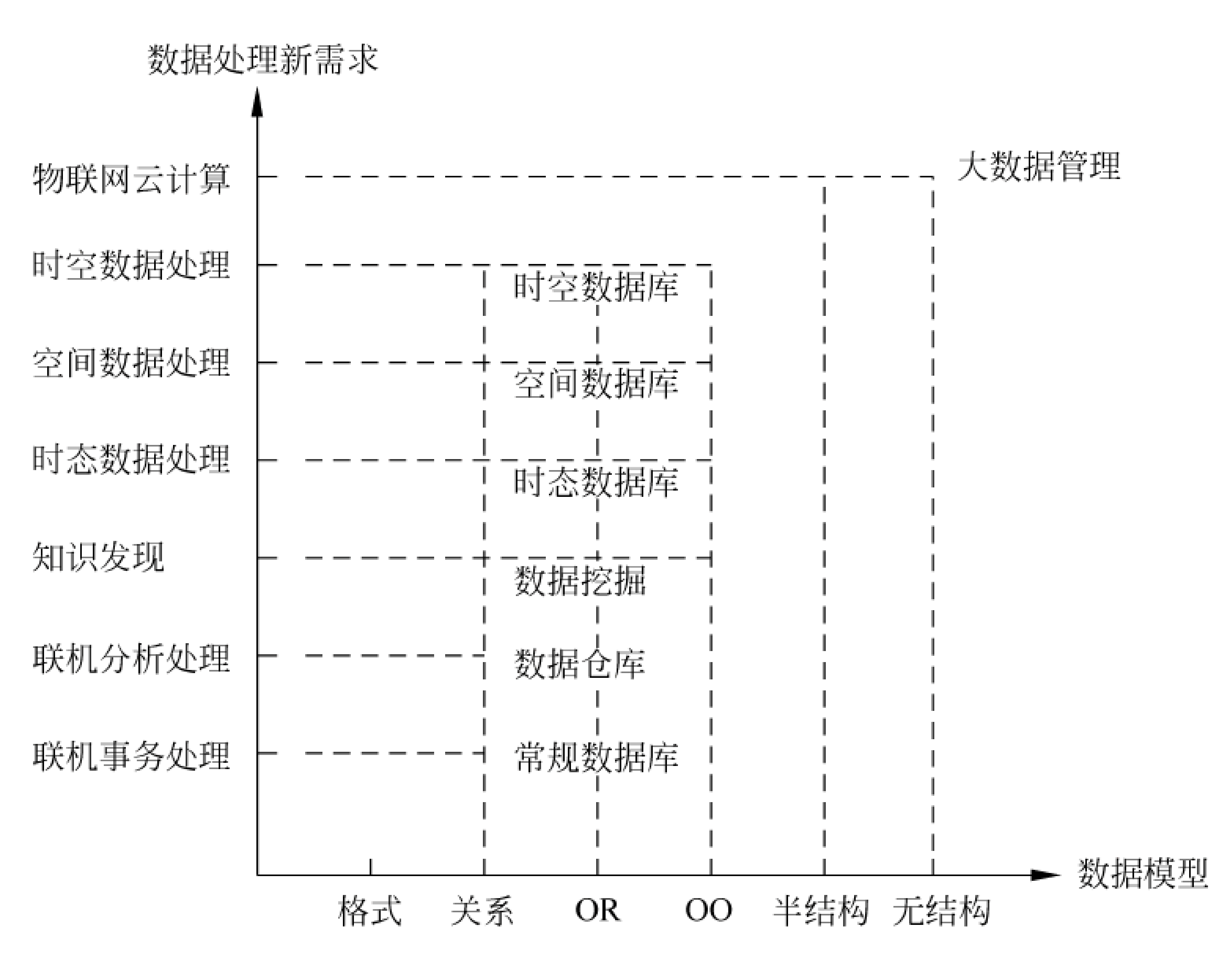


图 1-9 数据模型与数据处理新需求结合

1.3.2 数据库发展驱动要素

与一般客观事物相同，数据库技术的发展需要相应的条件和环境，这种条件和环境可以归结为数据自身、应用需求和硬件技术 3 个基本要素。

1. 数据自身变化

数据自身变化来源于计算机新应用领域产生的新型数据源。网络、科学研究、数字图书馆、电子商务和社交活动等已经成为数据和信息处理需求的重要数据源，其中的新型数据相对于传统情形发生了很大变化。

(1) 数据类型众多。数据库管理的基本数据对象长期以来主要是数字类型和字符类型。现在大量新型数据类型不断涌现，如图形图像数据、动画数据、音频视频数据、社交文本和多媒体数据、数据仓库中的多维数据、网络上的 HTML 和 XML 数据、时间序列数据、流数据和过程或行为数据等。

(2) 数据结构复杂。上述新型数据类型通常或者具有复杂数据结构，或者呈现半结构或无结构特征，或者存在结构但难以进行技术上的清晰描述。这种基于复杂数据结构建模比常规情形具有更大的挑战性，相应的数据操作、存储策略、存取方法和服务质量保障也相当复杂、困难。

(3) 海量数据出现。当今可以获取的数据量正在以 TB(Teta Bytes，1024GB) 和 PB (Peta Bytes，1024TB) 数量级增长。实际上，图灵奖获得者 J. Gray 早在 1999 年就预测"从现在起，每 18 个月全球新增信息量等于计算机有史以来存储量之和"。

海量数据主要来源于科学研究、物联网、电子商务和人们的文化社交活动。科学研究需要处理的数据量如表 1-3 所示。

表 1-3 当今科学研究前沿中需要处理的数据量

应用领域	应用方向	运算性能需求	存储容量需求
生物医学	蛋白质电子态计算	100Tfkoats	30TB
	药物发明中筛选	800Tfkoats	200TB
	蛋白质折叠	1Pfkoats	1PB
航空航天制造	发动机燃烧模拟和机翼设计模拟	500Tfkoats	100PB
气候环境	短期天气预报	20Tfkoats	10PB
	长期天气预报	200Tfkoats	100PB
	局部突发性灾难预报	1Pfkoats	500PB
核能技术	完全等离子分析(电子结构分析)	500Tfkoats	1PB
	核武器数值模拟	1Pfkoats	1PB
	天然气燃烧	1Pfkoats	1PB
纳米技术	符合材料结构分析与动能测试	200Tfkoats	400TB
	新材料发明	1Pfkoats	2PB
国家与国防安全	密码破译	1Pfkoats	1PB
	先进武器模拟	1Pfkoats	1PB
天体物理学	超新星三维模拟	1Pfkoats	1PB

电子商务和社交活动近年来由于无线通信终端设备的普及已经成为海量数据的主要贡献者。淘宝拥有 4.4 亿注册用户、8 亿多商品条目，每分钟销售 4.8 万件商品，高峰日成交

额达到 52 亿元，庞大的用户流、信息流和资金流，使淘宝积累起 20PB 规模以上的数据。Google 每天约有 200 亿个网页搜索；2014 年巴西世界杯中，现场超过 570 个视频传输设备，每天有超过 10 万个无线设备观看比赛，累计超过 10 亿人次通过互联网或无线移动设备观看全部 64 场比赛，一场比赛收集 6000 万条需要存储的相关记录等。据统计，世界每年产生的数据 92%存储在计算机磁盘和光介质当中，7%存储在胶片上，传统纸质印刷信息只占 0.01%左右。量的变化终究会导致质的改变，如何管理这些海量数据是数据库技术面临的新挑战。

数据自身的巨大变化为数据管理领域带来一系列挑战性课题。例如，是扩充传统 DBMS 功能，还是重新考虑其基本架构。又如，对这些结构化、半结构化和非结构化数据的混装体如何进行数据建模、数据存取和智能抽取等。这些问题涉及众多领域，更需要将数据库技术研究发展与计算机最新领域学科交叉融合，其中形成了以大数据为基本特征的新的数据管理研究与应用领域——数据科学。

2. 应用需求变化

当今，数据库应用领域呈现出多元化的明显特点。随着无处不在的数据管理需求，相应数据库的应用也无处不在，数据库系统已经成为计算机应用系统的核心与基础。

(1) 网络强劲推动。20 世纪 90 年代以来，Internet 已经成为最主要的应用领域，这个全新领域向数据库提出了前所未有的应用需求，电子商务和电子政务、信息服务和云计算、社交网络和智慧城市等新型数据库应用需求应运而生。例如，传统事务处理方式已经发生了重大变化，从企业或部门内部处理到以网络和 Internet 为基础，跨部门跨行业甚至全社会的开放处理，由此提出了对数据库信息安全和信息集成新的保障和支持需求。

(2) 科研提供需求。生物学、天文学、地理科学、保健科学和工程领域等科学技术研究领域产生大量复杂结构数据，这些数据的管理需要比当前数据库技术和产品所能提供的支持更为高级。同时也需要信息集成机制，对分析过程中的中间数据进行管理，需要和全球范围内的数据网格进行交互和集成。

(3) 应用环境开放。现在，数据库不再局限于一个机构内部的事务逻辑管理，而是处在一个面向开放具有更多需求的多元应用环境。分布自治的计算环境、移动设备的动态管理、实时处理和隐私保护等已经称为数据库重要的研究科目。开放式环境将促使数据库技术研究与开发的不断深入与发展。

新的计算机应用对于数据管理新的要求与挑战如表 1-4 所示。

表 1-4　新的应用领域对数据处理提出新的需求

应用领域	应用方向	运算性能需求	存储容量需求
生物医学	蛋白质电子态计算	100Tfkoats	30TB
	药物发明中筛选	800Tfkoats	200TB
	蛋白质折叠	1Pfkoats	1PB
航空航天制造	发动机燃烧模拟和机翼设计模拟	500Tfkoats	100PB
气候环境	短期天气预报	20Tfkoats	10PB
	长期天气预报	200Tfkoats	100PB
	局部突发性灾难预报	1Pfkoats	500PB

续表

应用领域	应用方向	运算性能需求	存储容量需求
核能技术	完全等离子分析(电子结构分析)	500Tfkoats	1PB
	核武器数值模拟	1Pfkoats	1PB
	天然气燃烧	1Pfkoats	1PB
纳米技术	符合材料结构分析与动能测试	200Tfkoats	400TB
	新材料发明	1Pfkoats	2PB
国家与国防安全	密码破译	1Pfkoats	1PB
	先进武器模拟	1Pfkoats	1PB
天体物理学	超新星三维模拟	1Pfkoats	1PB

3. 硬件技术变化

计算机硬件和相关技术成熟是数据库技术发展的另一动力。

1965 年,英特尔的创始人之一戈登·摩尔(Gordon Moore)在考察计算机硬件的发展过程之后提出了摩尔定律:同一面积芯片上可容纳的晶体管数量 1～2 年将增加 1 倍。而实际情况竟然与摩尔的预估惊人地相符。据统计,在 1971—2011 年的 40 年间,大概每两年相同面积的中央处理器集成电路上的晶体管数量就增加 1 倍。在 2014 年,英特尔公司宣布成功开发出 14nm 的 3D(三维)晶体管,由此大部分科学家相信,摩尔定律的生命将会一直延续到 2020 年。

摩尔定律带给人们的最大收益是计算机硬件价格的快速下降。与常理相悖的是,单位面积上晶体管数量的增加不仅没有引起集成电路价格的上涨,反而导致了快速下降,究其缘由是由于晶体管做得愈小其成本就愈低,同时相关设备的实际用户量也在成倍增加。

早在 2009 年,世界晶体管的产量就达到了世界大米颗粒数量的 250 倍,由此导致 1 粒大米的价钱可以购买 10 万个晶体管。1TB 容量硬盘价格也在持续下降,2012 年其价格为 94.99 美元,2014 年降至 49.99 美元,预计到 2020 年,1TB 容量硬盘的价格将降为 3 美元,这只相当于一杯咖啡的价格。从数据库管理技术发展的不同阶段可以看出,不同的硬件环境只能产生和实现相应水平的软件技术。如同所有计算机应用情形,数据库技术也依赖于硬件设备的发展,特别是处理器速度和存储器容量及其相应成本。计算机超级计算能力的提升和巨量存储设备的完善,为有效管理各类新型数据库提供了必需的技术支撑和发展的广阔空间。同时,计算机硬件的飞速发展正在改变着数据库技术中的已有规则,要求人们对传统数据库中的存储机制和查询方式重新予以评估,在新的计算机体系结构框架内重新设计相应的数据结构和基本算法。

1.4 数据库技术的地位和意义

世界上第一台真正意义上的电子计算机在 1946 年诞生于美国宾夕法尼亚大学的摩尔学院,正式名称为 Electronic Numerical Integrator and Computer,简称 ENIAC。ENIAC 当时主要用于弹道计算、火力表测试及科学研究。随着计算机理论研究的深入和计算机技术应用的发展,从 20 世纪 50 年代开始,计算机应用就开始由军事和科学研究领域逐渐扩展到行政部门和企事业单位。这种扩展的重要标志就是计算机主要应用范围由特定的面向数值

型数据的科学计算转变到一般的面向非数值型数据的数据管理和事务处理。伴随着这种转变的逐步推进，以数据统一管理为核心的数据库技术随之发展与成熟起来，成为计算机科学技术与应用中最为广泛和最为重要的领域之一。

数据库系统的出现是计算机理论和应用发展史上的里程碑之一，也是计算机应用领域从科学计算发展到事务与非事务处理历史性转变的基本标志之一。从计算机应用角度来说，正是由于这种转变，才使得计算机技术和应用得以深入各行各业和走进千家万户中发挥了重大作用；从逻辑上来看，在这个转变进程当中，数据库原理的深入研究和数据库技术的广泛应用有力地表明了计算机学科不仅仅只属于应用工程范围，也具有科学技术的内秉特质。数据库技术作为计算机领域的基本学科，已经成为整个计算机信息系统与应用领域的核心技术和重要基础之一，有着明显的学科地位和意义。

1.4.1 计算机领域中的学科地位

数据库技术是计算机领域的重要技术之一，也是计算机软件学科的一个独立分支，吸引着大量杰出人才献身其中，为计算机理论与技术发展做出了重要贡献。在数据库领域，被称为网络数据库之父的 C. W. Bachman，由于其主持设计与开发了最早的网络数据库系统 IDS，以及积极推动与促成了数据库标准的制定而获得了 1973 年的图灵(Turing)奖；被称为“关系数据库之父”的 E. F. Codd 提出关系模型，不仅为数据库技术的发展奠定了基础，同时也为计算机的普及应用提供了强大动力，因而获得了 1981 年的图灵奖；被称为“数据库技术和事务处理专家”的 James Gray，由于解决了数据库的完整性、安全性、并发性和故障恢复，使得数据库产品真正进入实用，被广大用户所接受，因而获得了 1998 年的图灵奖；被称为“现代数据库技术奠基者”的 M. Stonebreaker，由于在 Ingres 和 Postgres 中的破冰工作和在数据库领域一直扮演着思想领袖的角色获得了 2014 年图灵奖。

图灵奖通常被认为是计算机领域中的诺贝尔奖，此奖获得者都是计算机领域中极为杰出的专家、学者。计算机的飞速发展，造就了许多重要的分支领域，出现了不少顶尖人物，但在同一个学科领域中，持续有四位图灵奖得主，这在计算机科学技术其他领域并不多见，着实为人们所称道，并由此也可以体会到数据库在整个计算机领域中的学科地位和发展活力。

1. 图灵奖简介

杰出人物、最聪明大脑的加入，从来就是一门技术最重要的学科地位与领域价值的最显著标志。在计算机领域，衡量杰出人物和聪明大脑的通常标准也许可以看作是获得图灵奖与否。



图 1-10 Turing

图灵奖(A. M. Turing Award)被称为计算机领域中的诺贝尔奖，为纪念科学家 Turing(1912—1954，英国)(图 1-10)而设立。Turing 提出 Turing 机理论，并在“二战”时破解德国通信密码，挽救了无数生命；他提出仿真系统和自动程序设计概念，设计了“Turing 测试”。由于 Turing 对于计算机发展巨大的历史功绩，他被人们称为“计算机之父”“人工智能之父”和“密码破译之父”等。图灵奖由 ACM 于 1966 年设立，奖金 20 000 美元(1966—1988 年)，此后由于有关大企业的投资，逐步增加到 25 000 美元(1989—2002 年)、100 000 美元(2002—2006 年)、250 000 美元(2007—2014 年)和 1 000 000 美

元(2014 年至今,Google 公司赞助)。原则上每年奖励一名。图灵奖评选条件极高,评选程序极严。主要获得者为美国科学家,也有少数英国、瑞典和以色列科学家。图灵奖不仅发展中国家无人获得,就连计算机技术极为发达的日本、德国和法国等也无人获得。



图 1-11　姚期智

美籍华人姚期智(Andrew Chi-Chih Yao)(图 1-11),1946 年 12 月 24 日生于上海。1967 年获得台湾大学物理学士学位,1972 年获得美国哈佛大学物理博士学位,1975 年获得美国伊利诺依大学计算机科学博士学位。现任清华大学高等研究中心(The Center for Advanced Study in Tsinghua University)教授。由于在计算理论(包括伪随机数生成、密码学与通信复杂度)的诸多贡献,姚期智在 2000 年获图灵奖,成为迄今为止唯一一位获图灵奖的亚裔人士和美籍华人。

2. 数据库领域四位获奖者

C. W. Bachman(图 1-12),1973 年的图灵奖获得者。Bachman 最重要的贡献是 1964 年在通用电气公司主持设计与实现了网状数据库管理系统 IDS,其设计思想和实现技术成为后来许多数据库产品的重要参照。Bachman 还积极推动和促成了数据库标准的制定,并于 1971 年发布在著名的 DBTG 报告当中。DBTG 首次确定了数据库三层体系结构。由于 C. W. Bachman 做出的杰出贡献,其被尊称为“数据库先驱”和“网状数据库之父”。

E. F. Codd(图 1-13),1981 年图灵奖获得者。Codd 担任 IBM 圣约瑟研究实验室任高级研究员期间,1970 年 6 月在 *Communications of ACM* 上发表《大型共享数据库数据的关系模型》一文,提出了基于数学基础的关系数据模型,此篇论文被 ACM 列为 1958 年以来 25 年中最具里程碑意义的 25 篇论文之一。此后,Codd 继续致力于完善与发展关系数据理论。1972 年,提出了关系代数和关系演算的概念,定义了关系的并、交、投影、选择、连接等各种基本运算,为日后结构化查询语言(SQL)奠定了基础。经过 Codd 的系列工作,一批商品化关系数据库系统很快被开发出来并迅速占领市场。因而 Codd 被尊称为“关系数据库之父”。



图 1-12　C. W. Bachman

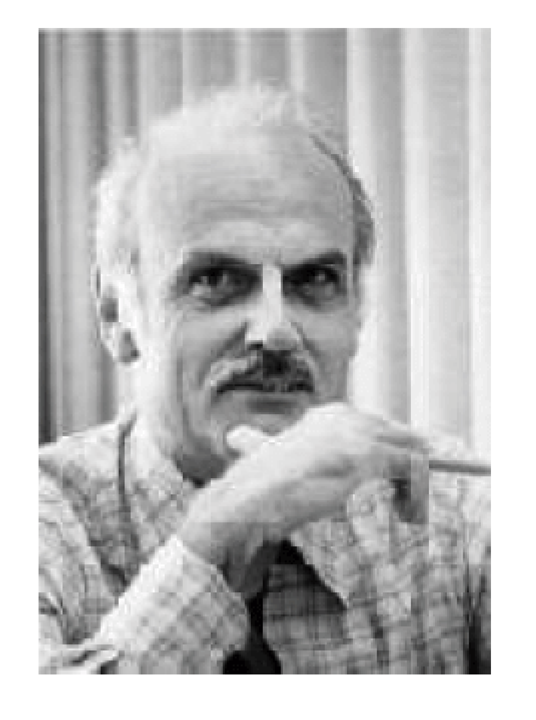


图 1-13　E. F. Codd

J. Gray(图 1-14),1998 年图灵奖获得者。在 IBM 期间,Gray 参与和主持 IMS、System R、SQL/DS、DB2 等项目开发,其中除 System R 作为研究原型外,其余已经成为 IBM 公司

在数据库市场上具影响力的产品。J. Gray 的杰出贡献是研究解决数据库事务处理技术。事务处理技术诞生于数据库研究后期，对于分布式系统、C/S 结构中的数据管理与通信和容错及高可靠性系统都具有重要意义，如果缺乏该项技术，任何数据库产品都无法进入实用而最终被用户接受。J. Gray 在 1993 年由 DEC 公司进入微软公司，微软公司专门为其在旧金山建立了第二个微软研究院“湾区研究中心”(Bay Area Research Center)，并任命他为研究中心主管。很快，Gray 带领的研制小组成功开发出 MS SQL Server 7.0，这是微软历史上具有里程碑意义的软件版本，该版本和其后续版本已经成为当今关系数据库市场上的领军系统。作为终身致力于通过数据库系统在人与人和人与物之间建立联系的超级天才，Gray 却在 2007 年 4 月 28 日驾游船出海后与所有的人永远地失去了联系。

Michael Stonebraker(图 1-15)，2014 年图灵奖获得者。作为现代数据库系统的概念和实践方面的奠基人之一，Stonebraker 的杰出贡献在于提出了促使数据库系统成为商业化产品的若干重要理念，而这些几乎都在所有现代数据库系统中得到应用实现。他在 Ingres 中引入了用于完整性限制和视图的查询修改，在 Postgres 中建立了对象-关系模型，有效地将数据库和抽象数据类型进行融合的同时，又保持了数据库和程序设计语言的分离。Stonebraker 还将 Ingres 和 Postgres 系统开源，促成了数据库的广泛应用，两者的代码库已被集成到各类现代数据库系统当中。Stonebraker 不仅拥有重大影响力的数据库技术成果，同时也在当今数据库领域一直扮演着思想领袖的角色，许多数据库领域中的领军人物都是其门下高足，如 Robert Epstein (Sybase 创始人)、Paula Hawthorn (Britton-Lee 创始人，曾任 Informix 研发副总)、Gerald Held (曾任 Oracle 研发副总)和 Margo Seltzer (哈佛大学教授，Berkeley DB 作者)等。

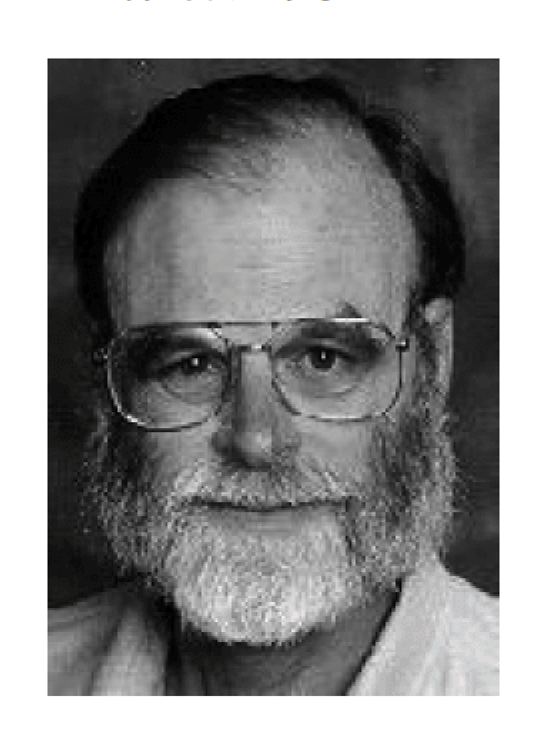


图 1-14　J. Gray



图 1-15　Michael Stonebraker

1.4.2　计算机应用领域的基础支撑

当今世界，计算机应用已经深入到社会的各个方面。数据库技术作为计算机学科最大的应用领域，不仅在传统商业等事务处理领域中发挥着极大的作用，而且在非传统应用中也起到越来越重要的支撑作用。事实上，任何一门具有重大意义的计算机应用，几乎都需要相应数据库技术的支撑。例如，在计算多媒体应用飞速发展的同时，多媒体数据库相应而生；与计算机网络同步，就有分布数据库和并行数据库技术；与互联网相呼应，就有 Web 数据库和 XML 数据库技术；与知识发现进展配合，就有数据仓库和数据挖掘技术；与知识的管理相匹配，就有知识库技术等；为物联网和云计算所推动，就有大数据管理(NoSQL 数据

库)技术;为 GPS 和无线位置服务所需求,就有移动对象数据库技术等。据人们统计,在所有计算机应用当中,专门将数据库技术应用(即使用了主流数据库平台及相关外围技术)作为专项技术进行研制开发,占了所有应用的 70%以上。

1.4.3 一个学科带动一个产业

由数据库技术发展进程可以看到,数据库学科发展和数据库软件产业发展紧密结合并相互促进。近 50 年以来,由于进行了较为完整和系统的基础理论研究,数据库技术具有坚实的理论支撑,形成一门研究范围广泛的理论与技术学科,其中主要包括数据库管理系统软件研制、数据库设计和数据库理论 3 个领域。数据库学科发展带动数据库软件产业的发展,实现了成功的发展链条,如图 1-16 所示。

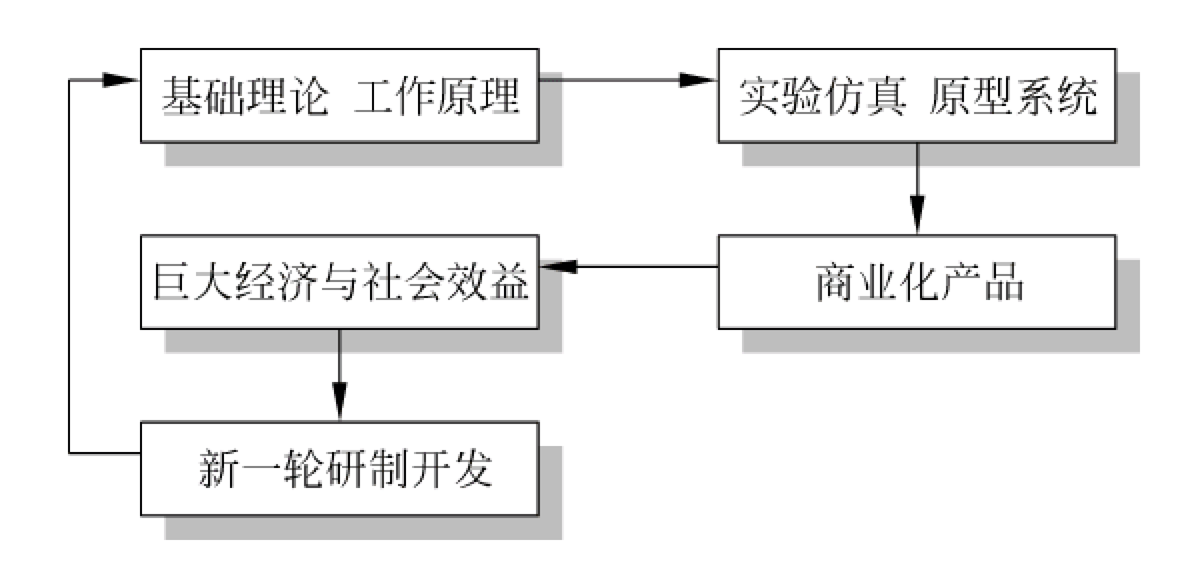


图 1-16 数据库技术发展链条

(1) 20 世纪 80 年代,数据库设计理论和设计方法学的研究首先在大学和研究所进行。以此为基础,研究部门和软件公司便着手研制数据库辅助设计系统。

(2) 20 世纪 90 年代,众多著名数据库公司逐渐将这些成果"拿"来进行产品化运作,推出各种数据库设计工具产品用以辅助大型数据库的设计,使之规范化和标准化。

(3) 如今,在全世界已有数百家数据库厂商,提供数百种 DBMS 核心产品、开发工具、设计工具、分析工具和解决方案,形成数百亿美元的数据库软件产业。

数据库的发展表明,对数据库理论与技术的研究投入带来了巨大的商业成功,产生了非常理想的收益回报。而且从投入到回报的循环只需短短十几年的时间,相关研究人员能够在有生之年亲眼看到这些成就所产生的巨大影响和经济效益,这无疑大大激励了研究者的研究动力和投资者(包括政府和公司)的资本投入,充分体现了"知识经济"和"信息时代"的时代特征。

一门学科带动一个社会产业,一种计算机应用系统带来巨大效益,现在数据库系统已经成为仅次于操作系统和办公软件的一类庞大的系统软件,奠定了自身在计算机科学技术与应用学科的基本地位。

1.4.4 保持强劲发展势头

从当前情况来看,计算机网络和多媒体技术已经成为信息技术的基本潮流。一方面,随着市场巨大需求及技术条件的日益成熟,基于网络的非传统复合类型数据(XML 数据、多媒体数据和社交媒体数据等)的存储、查询和处理,已经成为新世纪数据库应用的主流;另一方面,各类迅猛发展的计算机新技术与数据库技术相互融合与促进,也已成为数据库新技术本身发挥作用和保持强劲发展势头的重要前提之一。数据库可以看作是一门技术,而一

个技术能有如此旺盛的生命力，对计算机的最新发展有着极强的融合力，相对于一些技术方法兴盛衰落的情形，数据库始终紧随着计算机发展的大势潮头。

在计算机发展历史上，有不少基本重要的软件技术，由于硬件技术的发展和应用需求的改变而成为望眼浮云和匆匆过客。这是因为不少软件技术其根本都是基于相应的硬件水平，皮之不存，毛将焉附。数据库技术最初的提出当然也离不开当前的硬件发展程度，但数据库的基本思想是数据管理，其中的数据模型、查询更新、事务处理和安全性机制具有学科内涵和逻辑原理上的普适性质。因此，硬件技术的发展并不能作为结束其生命周期的“镰刀”，而是催促其生命更加勃发的“号角”，为数据库展示出更为广阔的发展前景。

计算机发展，往客匆匆，多少软件技术成为过眼烟云。然而，计算机技术（硬件和软件）越发展，计算机应用领域越显现出超强的适应性能，数据库技术就会在更为广阔的层面中和更为深入的程度上得到强化与推进。计算机领域各种新的进展与突破都为数据库技术开辟了新的发展空间与用武之地。

本 章 小 结

数据在计算机应用意义上可以看作是抽象符号的特定集合，其特征是需要在给定语境中表明其语义的。计算机数据分为数值型和非数值型两种情形。在计算机科学与技术领域当中，主要研究对象是非数值型数据。

在计算机应用环境中，数据的使用分为数据计算（数据处理）和数据管理。数据管理中的数据具有数据量巨大、数据持久保存和数据大范围共享的显著特征，其中最有效的管理技术就是数据库管理系统。

数据管理技术经历了基于应用程序（人工）管理、文件系统管理和数据库管理 3 个发展阶段；而数据库技术也经历了第一代数据库（格式化数据库：层次和网状数据库）、第二代数据库（关系数据库）和新一代数据库（第三代数据库，以对象数据库为主要代表）的发展历程。

由于计算机应用实际上就是存储、加工和使用数据，因此绝大多数计算机应用系统都需要数据库技术的底层支撑，数据库系统已经成为最大的计算机应用产业之一，具有完整的理论体系和广泛的应用领域，先后诞生了 4 位图灵奖获得者，并成功实现了技术研发与商业化产品的良好循环。

自 20 世纪 60 年代末 70 年代初期到现在，经过近半个世纪的发展，数据库已经形成数据模型丰富多彩、新鲜技术层出不穷和应用范围不断扩大深化的庞大家族。在这个家族中，由于各类技术的展开和系统研制都需要建立在数据模型基础之上，也就是说，所有类型的数据库技术都是基于或支持某种数据模型，因此数据模型就成为各类数据库技术的发展主线。按照数据模型考虑，可分为第一代（层次和网状模型）数据库、第二代（关系模型）数据库和新一代（以对象数据库模型为代表的各类新型数据模型）数据库。此外，为了具体厘清脉络，还可以从其他不同视角进行考虑。从与计算机涌现出的新技术结合角度，可以分为与分布式计算相关的分布式数据库、与 Web 技术相关的 Web 数据库、与人工智能相关的知识库和以多媒体技术相关的多媒体数据库；从时空维度角度考虑，有显式处理空间维度的空间数据库、显式处理时间维度的时态数据库，以及同时显式处理空间与时间维度的时空数据库，而将其他数据库类型看作“隐式”表示“当时当地”的快照数据库。此外，还可以从面向应用角

度考虑各类特定的数据库，如面向联机分析和知识发现领域的数据仓库和数据挖掘技术、面向 CPS 及无线通信网络终端智能应用的移动对象数据库等。从不同角度考察数据库技术的状况，实际上就是探究其发展的基本特征和未来趋势。

需要说明的是，从整体上来看，数据库技术的应用领域可以大致分为事务型应用和非事务型应用。从数据库技术的适应架构考虑，经典关系型数据库只适合于管理事务型数据，而各类新型的数据库应用需求大多都是基于非事务型数据。但是由于非事务型数据管理技术的复杂性与多样性，迄今为止乃至在可以预见的未来，研制开发出像关系数据库那样功能强大而又极其成功的情形可能还只是个例。面对大量的非事务数据管理的强劲需求，如前所述，要么直接将关系数据库应用其上，要么将关系数据库进行适当拓展（基于对象关系数据库的多媒体等），要么就是直接构造相应基本功能构件（如基于索引的查询与更新构件等）。由此看来，非事务型数据的管理是现代数据库技术的巨大挑战，当然也更是数据库技术取得更大发展的机遇。

主要参考文献

[1] Ling Liu and Tamer M. Encyclopedia of Database Systems[M]. Springer Science Business Media, LLC, 2009.

[2] 《数据库百科全书》编委会. 数据库百科全书[M]. 上海：上海交通大学出版社，2009.

[3] 王珊，等. 数据库与信息系统——研究与挑战[M]. 北京：高等教育出版社，2005.

[4] 孟晓峰，周龙骧，王珊. 数据库技术发展趋势[J]. 软件学报，2004 15(12)：1822-1836.

[5] 刘云生. 现代数据库技术[M]. 北京：国防工业出版社，2001.

[6] 汤庸，叶小平，等. 高级数据库技术与应用[M]. 北京：高等教育出版社，2015.

[7] 涂子沛. 数据之巅[M]. 北京：中信出版社，2014.

第 2 章　关系数据库基础

关系数据库(Relational DataBase,RDB)是基于关系数据模型的数据管理技术,它的原理和技术产生于 20 世纪 70 年代初期,发展于 20 世纪八九十年代。20 世纪 90 年代末期特别是进入 21 世纪之后,RDB 与计算机网络技术密切结合,通过各类扩充与发展,焕发出新的生命活力。由于 RDB 建立在数学理论之上,使得技术意义下的数据组织、管理和操作等具有较高的抽象层次和科学的理论特质,带动了整个数据库理论与技术的蓬勃发展,它的建立是数据库发展历史上最重要的事件。自 20 世纪 80 年代以来,各计算机厂商研制的数据库管理系统(DBMS)几乎都是关系类型,即便各种非关系系统也大都附加有关系接口。基于关系模型的 RDB 直到现在并且在今后相当长一段时间内,都是最重要和最流行的数据库。关系数据技术也是其他数据库原理和技术的基础支撑。本章通过简要回顾 RDB 的基本理论与技术,为高级数据库技术后续学习提供原理和技术上的准备。

2.1　关系数据模型

如前所述,数据管理是一类最大的计算机应用领域,其核心就是以统一管理数据和多用户共享数据为基本特征的数据库系统。数据模型不仅是数据结构和数据操作的抽象,还是数据管理的出发点,更是数据库原理探讨和技术研发的基础。数据模型的发展进程就是数据管理技术的演进轨迹。关系数据模型由静态的数据结构、动态的数据操作和语义限制的数据完整性约束三部分组成。

2.1.1　关系数据结构

关系数据结构本质上可以看作是一种数学结构。从数学上来看,关系数据结构是多个域集合上笛卡儿乘积的一个子集。在离散数学中,这样的子集通常称为域集合上的一个关系,这也是关系数据中“关系”一词的由来。

1. 关系和关系表

关系数据结构可以使用数学中的“关系”概念进行描述。

在数学中,给定 n 个集合 $D_1, D_2, \cdots, D_n$(这些集合可以有若干相同),可以定义其上的笛卡儿乘积为如下集合。

$$D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) \mid d_i \in D_i, i = 1, 2, \cdots, n\}$$

其中,元素 $(d_1, d_2, \cdots, d_n)$ 称为向量,$d_i (i=1,2,\cdots,n)$ 称为该向量的第 i 个分量。

笛卡儿乘积的一个子集就称为集合 $D_1, D_2, \cdots, D_n$ 的一个关系。

数学是科学的皇冠,具有最高层面的抽象意义。但就数学对于其他科学技术指导和应

用而言，并不是“螺丝刀和螺丝钉”之间那种“拿来就用”的简单关联。数学原理应用于具体技术领域的关键之一就是对抽象层面上的概念理论进行相关技术的“学科释义”，如数学图论中的“树”和“图”概念在多个技术领域中的广泛应用就是如此。在离散数学中，“关系”只是数学意义下的一个抽象概念，涵盖面极其广阔，具体到数据库的技术领域，需要对其进行基于数据管理框架内的技术性解读，或者说需要进行数学“关系”在数据库语境中的语义说明。这种语义解释实际上就是对“关系”概念进行各种具有明确技术实现指向的限制。基于数据库的数据管理最重要的语境就是数据模型，因此，首先就要对数学概念“关系”进行概念数据模型和逻辑数据模型的语义解读，其次还要进行其他必需的技术语义解读。

1）概念模型解释

将上述每个 $D_i(i=1,2,\cdots,n)$ 看作是给定数据值的集合也就是属性域（简称为“域”），并为每个域赋予一个具有语义的名称。“域”是一组具有相同数据类型的值的集合，如整数域、实数域和长度指定的字符串域等。这样就可以将 n 个域 $D_1,D_2,\cdots,D_n$ 上的笛卡儿乘积中 $D_1\times D_2\times\cdots\times D_n$ 的每一个向量元素看作是概念数据模型 E-R 图中一个实体，向量的每一个分量就是对应实体的一个“属性值”。同时将笛卡儿乘积中的给定子集即整个“关系”看作是相应实体构成的 E-R 模型中的实体集。

2）逻辑模型解释

在上述概念模型语义解释中，由于实体集中的每个实体都具有相同结构属性域，也就是具有相同的实体型。因此，可以将其中每个实体表示为“一行”，而将相应实体集表示为由实体“行”构成的一张二维平面表格。此时，如果再对每个属性域中的属性值加以“原子性”，即不可再分解的限制，并对实体集对应的平面表格添加一个由对应属性名构成的“首行”作为“表头”，则可将数学“关系”对应的“实体集”解释为一张平面表格。由于满足一定语法要求的表格与数学意义上的“关系”具有上述内在联系，因此通常将其称为“关系表”（relation table），关系表就是关系数据模型的逻辑结构。在不引起混淆的情况下，通常也将数据库语境中的“关系表”与带有数学印记的抽象概念“关系”有意混用，关系表就直接称为“关系”并简记为“R”。这种混用实际上就明确反映了关系数据结构具有坚实的数学支撑，从而基于关系数据结构的数据操作实际上也就可以归结为相应的数学运算。

3）关系数据结构的语义约束

上述定义的关系 R 需要满足 $R\subseteq D_1\times D_2\times\cdots\times D_n$，这是对关系 R 的基本语法要求。但这种基于数学意义的语法要求在某种意义上过于宽泛，同时又在其他意义下约束较严，还需要结合计算机自身特点和实际应用中需要，对于上述定义的关系 R 施加必要限制和解除某些原有约束。

（1）有限性限制。“域”和“关系”都是集合，数学中的集合可有“无限”个元素，而计算机只能处理“有限”个元素，因此，需要限定“域”和“关系”都只能包含有限个元素。

（2）原子性限制。数学意义下“属性域”中元素只要满足域集合给出的相应条件即可，但从关系数据管理技术实现角度考虑，却需要“域”中的元素（即“属性值”）不能有结构，应该具有“原子性”，这就是 RDB 中著名的 1NF 规范性限制。

（3）顺序性解除。“关系”由“行向量”组成，而数学意义上组成“向量”的各个分量之间的顺序是重要的，但在实际数据管理过程当中，按照人们实际应用中的直观经验，行的分量即属性值之间的顺序在很多情况下并无必要。例如，对于一个学生实体而言，是先讲“学号”

还是先讲“姓名”并不会影响对学生信息的了解。因此，关系表中组成“行”的属性值之间的顺序限制可以解除。

(4) 空值性拓展。数学意义下“行向量”的每个分量都需要取确定的“值”。关系表中的“元组”表示现实世界中的数据对象，其中的“属性值”表示数据对象的特征，而实际当中某些属性值却不能确定。例如，一个单位的职工关系表，其中的“住宅电话”属性值就可能不确定：有人必须填写，有人可填可不填而选择不填，还有人则没有安装等。因此关系表就可以解除“行向量”中“分量”必须取确定值的数学限制，允许某个或某些属性值“空置”以表明其不确定性，也就是说，关系表中属性值可以取“空值”，空值通常使用空缺符 NULL 表示。

从严格意义上来讲，满足上述各类约束和条件的“关系”才可以作为计算机语境中的“关系表”。此时，关系表中除表头行之外的每一行(数据记录)称为一个元组，每一列称为一个属性列，元组中的每一个数据项称为属性值。

此外，由上述数学“关系”概念的数据模型解读可知，关系数据结构与实体-联系概念数据模型(即 E-R 图)密切相关，再通过进一步的释义，就进入到逻辑数据模型的范畴，这实际上就为在 RDB 数据库设计中从概念模型到逻辑模型的转化原理和方法提供了内在的可行性和有效性支撑。

2. 主键和外键

由上述可知，关系数据结构中的元组表示为 E-R 模型中的“实体”，关系数据结构本身表示为实体集。注意到 E-R 模型是实体-联系模型，此时还需要讨论实体标识和实体集之间“联系”在关系数据结构的表示。这就是下述讨论的“主键”和“外键”。

1) 关系表中数据实体“标识”由“主键”实现

关系表中元组的唯一标识通常是其所具有的一组特定属性并称其为“超键”(super key)。

“最小”意义下的超键称为“候选键”(candidate key)。

一个关系表可有多个候选键，在应用中确定使用的候选键称为主键(primary key)，并规定主键列数据不能重复并不能为空。

2) 不同关系表间数据实体“联系”由“外键”实现

设 *A* 是关系表 R 和 S 的一个共同属性子集，如果 *A* 不是 R 的主键但是 S 的主键，则称 *A* 是 R(关于 S)的外键(foreign key)，并称 R 是参照关系(referencing relation)而 S 是被参照关系(referenced relation)。

设有下述 3 个关系表。

(1) 学生关系表 S：S(<u>Sno</u>,Sname,Ssex,Sage,Sdept)。

(2) 课程关系表 C：C(<u>Cno</u>,Cname,Cpno,Ccredit)。

(3) 学生课程关系表 SC：SC(<u>Sno,Cno</u>,Grade)。

其中，<u>Sno</u> 为 S 关系表中的主键，<u>Cno</u> 为 C 关系表中的主键，<u>Sno,Cno</u> 为 SC 关系表的复合主键；SC 关系表中的“<u>Sno</u>”需要是登记在册学生的学号，即在 S 表中已存在的学号；同时，SC 关系表中的“<u>Cno</u>”需要是已经开设课程的课程号，即在 C 表中已存在的课程号，这样，通过 SC 中属性“<u>Sno</u>”构成 SC 关于 S 的外键联系，通过 SC 中属性“<u>Cno</u>”构成 SC 关于 C 的外键联系，即 SC 中的两个(单元素)属性子集{<u>Sno</u>}和{<u>Cno</u>}分别是其关于 S 和关于 C 的外键。需要注意的是，单元素集{Sno}和单元素集{Cno}并不是 SC 的主键，只有双元素集{Sno,Cno}才构成 SC 的唯一主键，因此，SC 关于 S 和 C 的外键符合上述外键定义的要求。

关系表中一个或一组属性能否成为相应的主键和外键完全取决于相应的数据语义而不是语法规定，因此，主键和外键技术构成了关系数据基本的语义实现机制之一。

2.1.2 数据操作

数据管理的基本操作主要是查询和更新。由于关系数据结构是数学意义下的一种集合结构，关系数据的查询和更新操作也就可以归结为相应集合的数学运算，关系操作也因之称为关系运算。在数学中，命题逻辑和集合运算等价，在关系数据操作中，也有基于(集合)代数的关系运算与基于(命题)逻辑的关系运算之分，前者通常称为关系代数，后者称为关系演算。

按照所依据的数学基础不同，关系运算和建立其上的数据库语言可分为如下 3 种类型。

1) 基于关系代数

关系数据更新的实现途径如下。

(1) 将集合代数中集合的“并”运算解读为关系表中元组的插入操作。

(2) 将集合的“差”运算解读为关系表中元组的“删除”操作。

(3) 关系表中元组数据的“修改”则通过“先删除再插入”的技术方式予以实现。

关系数据查询由专门的数学中未曾出现过的新集合运算实现，这主要包括“选择”“投影”和“连接(广义笛卡儿乘积)”。这些基于关系数据查询的集合运算与并交差等经典集合运算在数学意义上基本一致，具有运算的封闭性。

关系代数中数据处理的基本单位或“变量”是元组。典型关系代数语言有 IBM United Kingdom 研究中心研制的 ISBL(Information System Base Language)，并将其应用于 PRTV (Peterlee Relational Test Vehicle)实验系统。

2) 基于关系演算

以数理逻辑中的谓词演算(主要是命题演算)为基础，按照基本处理单位即谓词变元对象不同而分为两种情形：以元组为变元的称为元组关系演算；以属性为变元的称为域关系演算。代表性的元组关系演算语言是 E. F. Codd 提出的 ALPHA，其中查询语句谓词为 GET，而更新语句谓词为 PUT、HOLD、UPDATE、DELETE、DROP。Ingres 所用的 QUEL 语言就是参照 ALPHA 语言研制的。典型的域关系演算语言是 M. M. Zloof 提出的 QBE (Query By Example)。QBE 以元组变量的分量(即域变量)作为谓词变元基本对象，其基本特征是基于屏幕表格进行数据查询，即以填写表格的方式构造查询，以示例元素(域变量)方式表达查询结果可能的情况，以表格形式显示相应查询结果。

3) 基于关系代数与关系演算整合

相应的数据库语言就是著名的关系数据查询语言 SQL。查询语言是系统提供给用户用以在数据库中获取查询结果的一种计算机语言，通常比一般程序设计语言具有更高的技术层次，这主要表现在查询语言可以分为过程化语言(procedual language)和非过程化语言(nonprocedual language)两种情形。前者在给出查询目标的同时需要给出具体操作步骤，而后者只需给出查询目标即可，具体实施步骤由系统自动选择和完成。

关系代数对应语言是过程化的，关系演算对应语言是非过程化的，它们都具有简洁明确和形式化的特点，同时也反映了由数据库提取数据的基本过程与技术，但由于缺少方便用户使用的商业化的“语法与语义修饰”，因而普及程度不够。SQL 整合了关系代数和关系演

算，集成了两者的优势，化解了其中的不足，形成了关系数据库的标准查询语言，当今几乎所有关系数据库都是通过SQL完成其数据查询与其他相关数据操作的。

2.1.3 完整性约束

如果一个关系表满足相应的基本语法要求，就可以认为是“合法的”，但“合法的”并不能自然就成为“合用的”，还需要进行必要的语义限制。前述就对“关系表”进行了一系列的基于数据管理的语义约束，这可以看作是从数据结构角度进行的语义限制。由于关系数据结构对应的概念数据模型E-R具有相当丰富的语义内容，仅有“结构”方面的语义限制还是不够，还需要对其数据操作过程进行必要的语义限制。通常将这种基于数据操作过程需要的语义限制称为数据的完整性约束。从逻辑角度考虑，这种完整性约束的基本要求就是限定数据对应实体集的“范围”（即“外延”）和描述数据语义之间联系。关系数据的完整性约束主要包括下述4个方面。

（1）实体完整性。数据的标识是进行数据语义处理的前提，因此首先需要保证在数据操作过程关系数据中元组的标识不能为空而且唯一，由此建立起关系数据的实体完整性约束：关系表中主键不能取空值并且不能重复。

（2）域完整性。域完整性是指列的值域的完整性，域完整性限制了某些属性中出现的值，把属性限制在一个有限的集合中。例如，如果属性类型是整数，那么它就不能是101.5或任何非整数。

（3）参照完整性。数据之间联系不仅是理解数据语义的基础，还可以通过研究数据联系以采用相应技术手段以提高数据处理效率，如通过具有联系的数据之间相互“引用”可达到减少存储空间和快速查找的良好效果。关系数据的参照完整性约束就是上述思想的技术实现手段。参照完整性：当属性集A是关系表R关于关系表S的外键时，A取空值或者取S中某个元组的主键值。这就意味着外键不能取被参照表中不存在的属性值。

（4）用户定义完整性。数据语义与数据所处语境有密切关联，实际用户的应用环境是最重要的数据语境之一。具体应用中的语义需要通过用户自身的约束来实现。关系数据的用户定义完整性约束是数据实际语境语义的具体实现，但由于与各种各样的应用背景关联，系统难以统一处理，呈现出复杂多样的特点。

2.1.4 关系数据模式

作为一个基本概念，数据模型是数据管理过程中数据结构、数据操作和完整性约束的完整抽象，从实际应用角度考虑，通常也将数据模式看作数据模型在一个单位部门中具体实现的内涵约定，而将数据实例看作其相应的外延实例。

1. 模式与实例

（1）数据模式：在给定数据模型框架内对具体应用数据的描述。数据模型可以类比于程序设计语言，数据模式可以看作是使用该语言的语法格式建立起来的一种面向应用的数据类型，数据实例可以看作是具有该数据类型变量的一个实际取值。按照一定的语法和语义约束，关系数据模型对应众多具有实际应用需求的关系数据模式，这就如同在C++中按照相应语法要求可以根据需要建立众多的“结构类型”或“数组类型”。实际数据管理过程中应用关系数据模型建立的具体关系数据库就是关系数据模式。

(2) 数据实例：相应数据模式在具体应用环境中的实现。例如，为了管理2017级计算机学院研究生数据信息，可以按照关系模型的语法和语义要求创建一个关系表(关系模式)，此时尚未填写具体数据的“空表”就是一个关系数据模式，它刻画了此时关系模式的构成方式，属于元数据层级；而由每个2017年入学研究生填写之后得到的就是关系数据实例，它记录了按照关系模式约定的实际数据信息，属于具体数据层级。

由此可知，关系模式就是关系模型在一个应用环境中的映射，而关系实例就是相应关系模式在具体数据上的实现。从技术实现形式来看，对于一个具体的关系表，表头就表示关系模式，而表头之后的各个具体元组就构成了关系实例。关系模式相对稳定，关系实例常常需要进行插入、删除和修改。

区分“关系模式”和“关系实例”的意义在于有效处理关系模型的“型”与“值”的问题。关系数据管理的关键在于关系模式的设计和管理。

需要注意的是，实际应用中通常并不刻意在语言表述上明确区分“关系模型”“关系模式”和“关系实例”，而将它们统称为“关系”，这是一种在严格明确其内涵之后的一种基于使用简洁的“有意混用”，这在前述和后续内容学习过程乃至其他学科中也常出现。

2. 三级模式/两级映射

RDB的基础是关系数据模型，关系数据模型在实际应用中会有各种各样不同的关系模式，而RDBMS(关系数据库管理系统)需要为这些基于关系数据模型的多样关系数据模式建立统一的管理机制，也就是关系数据模型的实际应用的技术实现机制，因而需要建立相应的数据库体系结构，这就是“三级模式/两级映射”的结构体系。

数据模型有逻辑数据模型和物理数据模型之分，相应的就有逻辑模式和物理模式之别。实际上，美国国家标准协会(ANSI)就将数据模式分为内模式(物理模式)、概念模式(逻辑模式)和外模式(用户模式)3种级别。

1) 三级模式/两级映射体系结构

关系数据库体系结构都是建立在三级模式/两级映射的原理基础之上的。

(1) 内模式。使用物理数据模型对一个应用单位所涉及数据的描述，内模式对于一般用户透明，主要由相应的DBMS(数据库管理系统)设计处理。

(2) 概念模式。使用逻辑数据模型对一个应用单位所涉及数据的描述，这是数据库设计的最重要工作，通常所说的数据库模式设计就是数据库概念模式的设计。

(3) 外模式。对一个应用单位中某个或某些用户所涉及数据的个别描述，外模式可以看作是相应概念模式的一个子集，需要从相应概念模式推演而得。

数据库系统的上述三级模式结构的意义在于，将一个应用单位所涉及数据的技术组织交由DBMS完成，使得用户不必具体处理数据在计算机中的表示和存储方式，专注在较高的抽象层面上对数据进行有效管理。

上述3个数据模式可以通过两级映射进行相互转换：外模式/概念模式映射；概念模式/内模式映射。

2) 数据独立性

如前所述，数据独立性是数据共享性的基本保障之一，而“三级模式/两级映射”体系结构的意义就在于从技术实现上保证数据库中的数据独立性。

(1) 外模式/概念模式映射。外模式/概念模式映射保证了数据的逻辑独立性：即当概

念模式发生变化时，如添加新的关系表、取消原有关系表，或者对已有关系表增添新的数据或删除原有属性，只需要改动相应的映射，而外模式本身并不需要改变。用户的应用程序依据相应外模式编写，所以应用程序也无须改变。

(2) 概念模式/内模式映射。概念模式/内模式映射保证了数据的物理独立性，即当内模式由于采用了新的存储结构(如原先无索引需要改用索引存储结构等)而发生改变时，只需要改动相应的映射，而概念模式本身也不需要改变，进而对应的外模式也无须改变；反之也是如此。

在实际应用过程中，相对于结构本身的内在变动，结构之间映射的变动更为便捷和易于实现，上述体系结构基本点正是有鉴于此。

由于三级模式结构使得数据在计算机内的表述与组织对于用户透明，两级映射提供了数据的逻辑与物理独立性，因此三级模式/两级映射就成为现今所有数据模式的基本架构，也称为数据库的三级模式/两级映射的体系结构。

2.2 关系数据库标准语言

SQL(Structure Query Language，结构化查询语言)由 Boyce 和 Chamberlin 在 1974 年提出，并在 IBM 开发的关系数据库管理系统原型 System R 中予以实现。SQL 已发展成为一个通用性功能极其强大的关系数据库标准语言，几乎所有关系数据库都支持 SQL。

2.2.1 SQL 发展与基本功能

从根本上来看，计算机的基本功能就是进行数据处理和数据管理。数据管理主要是通过查询技术对数据进行频繁访问。人们使用数据库的基本目的是高效、正确和完整地查询所需要的数据，因此，数据库的核心技术就是数据查询技术，由此也就决定了数据库语言本身不同于一般高级程序设计语言的特征。SQL 正是这样一种能够实行有效查询而得到广泛使用的关系数据库标准语言。在 SQL 发展历程中，最具有里程碑意义的是 SQL-92 和 SQL-99，前者集经典关系数据管理之大成，后者开启了由经典关系数据管理到新型数据管理的历史性转变。

SQL 具有下述基本特点。

(1) 非过程操作。先前各种数据库语言与常规编程语言一样，用户需要着力描述一项操作的具体实现过程，即不仅告知系统“做什么”，更要告知“如何做”，因此是一种导航式的面向过程语言；而 SQL 就完全不同，只要告知系统“做什么”，具体实现(如存取路径和数据操作过程等)完全对用户透明，因此是一种面向结果的“非过程”语言，大大拓展了关系数据库的用户面。

(2) 面向集合处理。各类程序语言的每次处理的对象大都是基于单个数据，即“一次一数据”；而 SQL 每次处理对象是“元组”的集合即关系表，同时输出结果也是关系表，即“一次一集合”，从而能够简化相应的系统设计，增强系统效率。

(3) 多样使用方式。能够独立地用于数据处理的联机交互，即以通过单独使用 SQL 完成指定操作的方式供数据库一般用户使用，同时也能将 SQL 语句嵌入在其他高级程序设计语言(如 C++)当中供程序设计人员使用。SQL 以一种统一的语法结构而提供多种应用方

式的做法为实际应用带来极大的灵活性与方便性。

(4) 语言简洁易学。为了实现创建、查询和更新等关系数据库的核心操作,SQL只需要9个动词即可,各类SQL语句十分接近于自然英语表达,简洁易懂,如表2-1所示。

表 2-1 SQL中动词关键字

基本功能	相应动词	基本功能	相应动词
模式定义	CREATE,DROP,ALTER	数据操作	INSERT,DELETE,UPDATE
数据查询	SELECT	数据控制	GRANT,REVOKE

(5) 高度综合统一。前述ISBL、ALPHA和QBE主要具有查询和更新机制,但SQL却具有完成数据库几乎所有功能的综合统一机制,如数据库的重构与维护,完整性与安全性,并发控制与故障恢复等。

SQL是迄今为止最为成功的数据库语言,被称为"星际间的交流语言"。但如前所述,SQL本质上是基于数据管理而非数据计算处理的,因而是一种计算非完备语言,不能独立进行应用程序的编制与运行。如果进行一般数据处理,需要将SQL语句嵌入到常用的高级程序设计语言(如C++等)当中。这既是SQL的缺点,也是SQL的优点,因为将数据管理从数据处理计算中分离出来,就如同将非数值型数据从数值型数据中分离出来一样,是计算机应用观念和技术上的一大进步,因为两者具有各不相同的技术特征与应用需求。

SQL称"结构化数据查询语言",这只是一种理论上的逻辑界定。在实际操作过程中,为了正确、完整和有效地实施数据查询还需要其他重要机制的保驾护航。例如,为了保证数据的正确性就需要数据的完整性和安全性保护机制,为了得到数据的完整性和一致性就需要数据模式的规范化机制,为了保证数据共享性的有效实现就需要数据的并发处理和故障恢复等事务管理机制。而所有这些数据查询的"辅助保障"功能都可以由一种统一的SQL语言予以实现,这种情形是其他任何数据库操作语言所难以比肩的,也是SQL重要价值与基本意义的最显著体现。

2.2.2 关系定义

SQL中关系定义主要是数据模式定义,包括下述基本情形。

(1) 定义数据库模式——数据库。

(2) 定义关系模式——基本表。

(3) 定义外模式——视图。

需要注意的是,此时的"定义"实际上包括"创建"(CREATE)、"撤销"(DROP)和"修改"(ALTER)3项内容。

为了说明SQL数据操作的基本功能,建立如前已述的学生课程数据库模式S_C。

(1) 学生关系表S:S(Sno,Sname,Ssex,Sage,Sdept)。

(2) 课程关系表C:Course(Cno,Cname,Cpno,Ccredit)。

(3) 学生课程关系表SC:SC(Sno,Cno,Grade)。

1. 数据库模式定义

在SQL标准中,定义数据库(模式)通过CREATE DATABASE语句实现。

【例 2-1】 创建学生-课程数据库S_C。

```
CREATE DATABASE S_C;
```

说明：上述语句定义了数据库 S_C。

【例 2-2】 撤销学生-课程数据库 S_C。

```
DROP DATABASE S_C CASCADE;
```

说明：上述语句撤销即删除数据库 S_C。其中，CASCADE 表示该数据库中定义的关系表也同时被删除。数据库撤销语句的一般形式为：

```
DROP DATABASE <数据库名> < CASCADE|RESTRICT >
```

说明：CASCADE（级联）表示撤销数据库模式的同时把其中所有的数据库对象全部删除；而 RESTRICT（限制）表示如果数据库中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行，即只有当数据库中没有任何下属的对象时才能执行撤销。

2. 基本表定义

【例 2-3】 建立"学生"关系表 S，其中主键为学号，姓名取值唯一。

```
CREATE TABLE S
 (Sno CHAR(9) PRIMARY KEY,        /* 列级完整性约束条件 */
  Sname CHAR(20) UNIQUE,          /* Sname 取唯一值 */
  Ssex CHAR(2),
  Sage SMALLINT,
  Sdept CHAR(20)
  );
```

【例 2-4】 建立一个"课程"关系表 C。

```
CREATE TABLE C
   (Cno CHAR(4) PRIMARY KEY,
   Cname CHAR(40),
   Cpno CHAR(4),
   Ccredit SMALLINT,
   FOREIGN KEY (Cpno) REFERENCES C (Cno)
   /* 表级约束条件，自连接，Cpno 是外键，被参考表是本表 C */
   );
```

【例 2-5】 建立一个"学生选课"关系表 SC。

```
CREATE TABLE SC
   (Sno CHAR(9),
   Cno CHAR(4),
   Grade SMALLINT,
   PRIMARY KEY (Sno,Cno),
            /* 主键由两个属性构成，必须作为表级完整性进行定义 */
   FOREIGN KEY (Sno) REFERENCES S (Sno),
            /* 表级完整性约束条件，Sno 是外键，被参照表是 S */
   FOREIGN KEY (Cno) REFERENCES C (Cno)
            /* 表级完整性约束条件，Cno 是外键，被参照表是 C */
   );
```

说明：定义关系表时可能需要同时定义完整性约束。在 SQL 中，如果一个约束只针对

一个属性，可以在定义该属性的同时定义相应约束，此时称为列级完整性约束，一般情况下，非空约束总是列级约束；如果一个约束涉及多个属性，需要在整个属性定义之后再定义相应约束，此时称为表级完整性约束。

3. 视图定义

【例 2-6】 建立计算机科学系的学生视图。

```
CREATE VIEW CS_S
AS
     SELECT Sno,Sname,Sage
     FROM Student
     WHERE Sdept = 'IS';
```

如需删除视图 CS_S：

```
DROP VIEW CS_S;
```

2.2.3 数据查询

关系数据查询分为单表查询、基于连接的多表查询、基于嵌套的多表查询 3 种基本情形。

1. 单表查询

【例 2-7】 查询计算机科学系(CS)全体学生的名单。

```
SELECT Sname
FROM S
WHERE Sdept = 'CS';
```

说明：这是一个单表查询，其特征是 FROM 子句后面只跟有一个表名。

上述语句称为一个查询模块，也称为 SELECT 语句，为了便于理解，可以认为其中 SELECT 子句对应关系代数中的投影运算，FROM 子句对应连接运算(当涉及多个表时)，WHERE 子句对应选择运算。关键字 WHERE 之后通常是一个谓词表达式，SQL 标准中主要的谓词如表 2-2 所示。

表 2-2 SQL 中基本谓词

查询条件	对应谓词
比较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空值	IS NULL，IS NOT NULL
多重条件(逻辑运算)	AND，OR，NOT

2. 基于连接的多表查询

【例 2-8】 查询每个学生及其选修课程的情况。

```
SELECT Student. * ,SC. *
FROM   Student JOIN SC ON Student.Sno  =  SC.Sno;
```

说明：上述语句实际上是一个等值连接查询，如果需要实现自然连接查询，则可改写为：

```
SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
FROM   Student NATURAL JOIN SC;
```

说明：自然连接是在两张表中寻找那些数据类型和列名都相同的字段，然后自动地将它们连接起来，并返回所有符合条件的结果。

3. 基于嵌套的多表查询

嵌套查询分为不相关子查询和相关子查询两种方式。

【例 2-9】 查询选修了课程名为"DB"的学生学号和姓名。

```
SELECT Sno,Sname          /* ③ 最后在 Student 关系中取出 Sno 和 Sname */
FROM Student
WHERE Sno IN
     (SELECT Sno          /* ② 然后在 SC 关系中找出选修了 DB 课程号的学生学号 */
      FROM   SC
      WHERE Cno IN
           (SELECT Cno    /* ① 首先在 C 中找出"DB"的课程号 */
            FROM C
            WHERE Cname = 'DB'
           )
     );
```

说明：首先这是一个三层不相关多行子查询，其特征是内层查询不依赖外层查询，是一个可以单独运行的查询模块，系统是"由内向外"执行相应操作。其次，每一个子查询可能会返回多行结果，所以条件表达时只能用多行集合关键字"IN"或"not IN"，而不能用单行运算符"＝"或"！＝"。

【例 2-10】 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno,Cno
FROM   SC   x
WHERE Grade >= (SELECT AVG(Grade)
                FROM   SC y
                WHERE y.Sno = x.Sno);
```

说明：这是一个相关单行子查询，其中内层查询依赖于外层查询中涉及的关系表。

一般来说，如果一个多表查询的最终输出结果涉及多个表的属性列，此时，只能使用连接查询而不能使用嵌套查询，这是因为嵌套查询中子查询模块必须是单列查询。一般而言，通常嵌套查询也可以由连接查询实现，但不是所有连接查询都可由嵌套查询实现。

【例 2-11】 设有如下嵌套查询：

```
SELECT Sname
FROM Student
WHERE EXISTS
     (SELECT *
      FROM SC
      WHERE Sno = Student.Sno AND Cno = '1'
     );
```

还可以转换为如下连接查询：

```
SELECT Sname
FROM Student,SC
WHERE Student.Sno = SC.Sno AND SC.Cno = '1';
```

2.2.4 数据更新

数据更新分为数据插入、数据删除和数据修改 3 种情形。

1. 数据插入

SQL 中有插入元组和插入子查询结果两种方式。

【例 2-12】 将一个新学生元组(学号：201215128；姓名：Raul；性别：male；所在系：CS；年龄：18 岁)插入到 S 表中。

```
INSERT
INTO S (Sno,Sname,Ssex,Sdept,Sage)
VALUES ('201215128','Raul','male','CS',18);
```

【例 2-13】 对每一个系，求学生的平均年龄，并把结果存入数据库。

首先，定义相应关系表 Dept_age。

```
CREATE TABLE Dept_age
    (Sdept CHAR(15)        /* 系名 */
    Avg_age SMALLINT       /* 学生平均年龄 */
    );
```

其次，向 Dept_age 插入数据。

```
INSERT
INTO Dept_age(Sdept,Avg_age)
   SELECT Sdept,AVG(Sage)
   FROM Student
   GROUP BY Sdept;
```

说明：在插入子查询结果时，要求子查询返回的结果列与目标列从左至右在数量和数据类型上一一对应，可以有多行。

2. 数据删除

【例 2-14】 删除 CS 中所有学生的选课记录。

```
   DELETE
   FROM SC
   WHERE 'CS' = (SELETE Sdept
                 FROM Student
                 WHERE Student.Sno = SC.Sno
                );
DELETE
   FROM SC
      WHERE Sno IN (SELETE Sno
                    FROM Student
                    WHERE Sdept = 'CS')
```

3. 数据修改

【例 2-15】 将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET Grade = 0
WHERE 'CS' = (SELETE Sdept
             FROM Student
             WHERE Student.Sno = SC.Sno
            );
UPDATE SC
SET Grade = 0
WHERE Sno IN (SELETE Sno
              FROM Student
                    WHERE Sdept = 'CS')
```

2.3 关系模式设计

如前所述,关系数据模式设计就是关系数据库逻辑设计,主要是针对具体问题,如何构造一个适合于它的数据模式,这就需要建立关系模式设计的规范化理论。

在实际应用问题中,需要将客观实体的各种基本特征抽象为多种属性,然后再将这些属性构成相应关系模式。实际中抽象出来的多种属性不能简单地凑成几个关系模式,不能将有关联和没有关联的,以及有"强关联"和"弱关联"的不同属性随意组成关系表,这样会带来数据冗余,而冗余会引发数据异常。因此,需要描述和研究所得到属性之间的关联,这种关联就是数据依赖。数据依赖是关系模式内部属性之间的约束关系,作为现实世界中实体特征相互间联系的抽象,它反映了数据的内在性质,是数据语义的实际体现。数据依赖主要有函数依赖(Functional Dependency,FD)、多值依赖(Multivalued Dependency,MVD)和连接依赖(Join Dependency,JD)。

【例 2-16】 高校教务管理的应用中,需要描述"学生学号"(Sno)、"所在系"(Sdept)、"系主任姓名"(Mname)、"课程名"(Cname)和"成绩"(Grade)等属性,可以将其简单地组成一个关系模式 Student $<U,F>$。

属性组: U = {Sno,Sdept,Mname,Cname,Grade}

U 上函数依赖 F: $F=\{Sno \to Sdept, Sdept \to Mname, (Sno,Cname) \to Grade\}$,此时就会产生大量数据冗余,由此将会导致插入异常(Insertion Anomalies)、删除异常(Deletion Anomalies)和更新异常(Update Anomalies),导致数据不一致性。

而关系模式设计(即规范化理论)正是用来改造关系模式,通过分解关系模式来消除其中不合适的数据依赖,以解决插入异常、删除异常、更新异常和数据冗余问题。

2.3.1 函数依赖

函数依赖是最基本的数据依赖。

函数依赖:设 $R(U)$是一个属性集 U 上的关系模式,X 和 Y 是 U 的子集。若对于$R(U)$的任意一个可能的关系 r,r 中不可能存在两个元组在 X 上的属性值相等,而在 Y 上的属性值不等,则称"X 函数确定 Y"或"Y 函数依赖于 X",记作 $X \to Y$。其中 X 称为决定因素属性

组，Y 称为依赖因素属性组。

函数依赖可以分为平凡与非平凡、部分与完全及传递与直接函数依赖 3 种情形。

1. 平凡函数依赖与非平凡函数依赖

如果 $X \to Y$，但 $Y \subseteq X$，则称 $X \to Y$ 是非平凡的函数依赖；若 $X \to Y$，但 $Y \subseteq X$，则称 $X \to Y$ 是平凡的函数依赖。

【例 2-17】 在关系 SC(Sno,Cno,Grade)中有以下两种情形。

(1) 平凡函数依赖：(Sno,Cno)→Sno(Sno,Cno)→Cno。

(2) 非平凡函数依赖：(Sno,Cno)→Grade。

2. 完全函数依赖与部分函数依赖

在 $R(U)$中，如果 $X \to Y$，并且对于 X 的任何一个真子集 X'，都有 $X' \nrightarrow Y$，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{F} Y$。

若 $X \to Y$，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作 $X \xrightarrow{P} Y$。

【例 2-18】 在例 2-17 中，(Sno,Cno)→Grade 是完全函数依赖，(Sno,Cno)→Sdept 是部分函数依赖，这是因为 Sno→Sdept 成立，且 Sno 是(Sno,Cno)的真子集。

3. 传递函数依赖与直接函数依赖

在 $R(U)$中，如 $X \to Y$，($Y \subseteq X$)，$Y \to XY \to Z$，则称 Z 对 X 传递函数依赖。记作：$X \xrightarrow{\text{传递}} Z$，否则，就称为 Z 对 X 直接依赖。

注意： 如果 $Y \to X$，即 $X \longleftrightarrow Y$，则 Z 直接依赖于 X。

【例 2-19】 在关系 Std(Sno,Sdept,Mname)中，有 Sno→Sdept，Sdept→Mname，则 Mname 传递函数依赖于 Sno。

2.3.2 公理系统及有效性和完备性

如前所述，“平凡依赖”“部分依赖”和“传递依赖”是产生数据冗余的主要原因，因此，消除数据冗余的途径就是找出给定属性集合上的所有函数依赖，对其中能够产生数据冗余的函数依赖进行适当处理。由函数依赖概念给定属性集合上的函数依赖个数与该属性集合的幂集相关，这样就需要使用计算机“计算”函数依赖。函数依赖是语义概念，计算机能够处理的问题实际上是基于语法的，需要将函数依赖的语义问题转化为相应语法问题才能交由计算机处理，然后再将处理后的语法问题转回语义问题供用户使用。这里需讨论下述课题。

① 构建一种语法的公理系统以完成语义和语法相互转换。

② 证明这种“语义”和“语法”的来回转化是否“等价”，也就是说语义转为语法，语法再转回语义得到的是否就为原来意义下所需要的结果。

显然，这不仅是在关系数据理论范畴，就是在整个计算机科学技术领域都具有意义和充满挑战。幸运的是，人们在关系数据库中较好地解决并处理上述课题，形成了关系数据模式设计规范化的重要理论基础。

针对①，人们建立了著名的 Armstrong 公理系统；针对②，人们严格证明了 Armstrong 公理系统的有效性和完备性。

1. 函数依赖集的闭包

研究函数依赖是解决数据冗余的重要课题，其中就是要在 $R(U)$中找出其函数依赖。

对于给定关系模式 $R(U)$，理论上总有函数依赖存在，如平凡函数依赖和由候选键确定的函数依赖。因此，人们通常会比较容易地指定一些语义明显的函数依赖以构建一个函数依赖集合 F，以 F 作为讨论 $R(U)$ 上“所有”函数依赖的初始基础。因此，需要研究如何通过已知初始函数依赖集合 F 得到其他未知函数依赖。

【例 2-20】 设有关系模式 $R(U)$，X、Y、$Z \subseteq U$，A、$B \in U$。已知 $X \to \{A,B\}$、$X \to Y$ 和 $Y \to Z$ 是 $R(U)$ 上非平凡函数依赖。按照函数依赖概念可得函数依赖 $X \to \{A\}$ 和 $X \to \{B\}$；按照传递依赖概念可得函数依赖 $X \to Z$。此时，函数依赖 $X \to \{A\}$、$X \to \{B\}$ 和 $X \to Z$ 并不直接显现在问题当中，而是按照一定规则（函数依赖和传递函数依赖概念）由已知函数依赖“推导”出来。将此一般化，就是如何由已知的函数依赖集合 F，推导出新的函数依赖。

为了表述简洁和推理方便，对有关记号使用做如下约定。

(1) 如果 X、Y 等是 U 的属性子集，并集 $X \cup Y$ 简记为 XY。

(2) 如果 A、B 等是 U 中的属性，则集合 $\{A,B\}$ 简记为 AB。

(3) 如果 X 是属性集，A 是属性，将并集 $X \cup \{A\}$ 简记为 XA 或 AX。

以上针对两个对象情形，对于多个对象情形也做类似约定。

(4) 在给定初始函数依赖集合 F 时，关系模式 $R(U)$ 根据需要有时记为 $R(U,F)$。

下面先说明由函数依赖集 F“推导”出函数依赖的确切含义。

① 函数依赖集合 F 的逻辑蕴含。设有关系模式 $R(U,F)$，X、$Y \subseteq U$，若 R 中每个满足 F 中函数依赖的关系实例 r 也满足 $X \to Y$，则称 F 逻辑蕴含 $X \to Y$，记为 $F \models X \to Y$。

考虑到 F 所蕴含的所有函数依赖，就得到函数依赖集合闭包的概念。

② 函数依赖集合 F 的闭包。设 F 是函数依赖集合，被 F 逻辑蕴含的函数依赖的全体构成的集合，称为函数依赖集 F 的闭包（Closure），记为 F^+，即

$$F^+ = \{X \to Y \mid F \models X \to Y\}$$

显然有 $F \subseteq F^+$。如果还有 $F = F^+$，则称 F 是函数依赖的完备集合。

按照上述定义，由已知函数依赖集 F 求得新函数依赖问题可以归结为求 F 的闭包 F^+。但根据函数依赖定义完成这项工作却相当困难，这主要是因为属性间函数依赖关系存在与否完全取决于数据的语义。例如，对于一个教师来说，如果只允许有一个电子邮箱，则教师工号确定后，其电子邮箱地址也随之确定，即电子邮箱地址函数依赖于教师工号。但如果教师有多个电子邮箱，则上述函数依赖就不存在。确定属性间函数依赖，需要仔细研究数据语义，不能仅仅根据当前数据值进行归纳，更不能“想当然”。语义问题涉及问题较多，在实际情况中，人们难以从语义方面得到所需要的各种新的函数依赖，更难以保证没有遗漏地得到能由 F 所逻辑蕴含的“所有”函数依赖。

数理逻辑提供了解决问题的思路，那就是先将语义求解考虑转换到语法求解机制上去，将“逻辑蕴含”转换为“逻辑推导”，然后再讨论两者的“等价性”，即通常所说的有效性与完备性。这种基本考虑在关系模式设计理论中就体现为基于 Armstrong 公理系统的函数依赖推导原理。

2. Armstrong 公理系统

为了建立基于函数依赖的语法系统，从而求得已知函数依赖集合 F 的闭包 F^+，W. W. Armstrong 于 1974 年提出了一套推导规则。使用这套规则，可以由已有函数依赖“逻辑推导”出新的函数依赖。后来经过不断完善，形成了著名的“Armstrong 公理系统”，为关系模

式设计提供了一个有效并且完备的理论基础。

1）基本公理与推理规则

在下面描述的形式系统中，诸如 $X \rightarrow Y$ 之类的函数依赖公式都看作“形式公式”，这种形式公式和基于公理系统的形式推导（逻辑推导）在数理逻辑中比较常见。

（1）基本公理。Armstrong 公理系统有 3 条基本公理（推理规则）。

① A_1（自反律，reflexivity）：如果 $Y \subseteq X \subseteq U$，则 $X \rightarrow Y$。

② A_2（增广律，augmentation）：如果 $X \rightarrow Y$ 在 $R(U)$ 上成立，且 $Z \subseteq U$，则 $XZ \rightarrow YZ$。

③ A_3（传递律，transitivity）：如果 $X \rightarrow Y$ 和 $Y \rightarrow Z$ 成立，则 $X \rightarrow Z$。

作为一个公理系统，还应有相应的推理规则和公式（递归）定义，本书不讨论这些内容。下面只需应用一些基本的推理规则，并将给定的初始函数依赖集合 F 的函数依赖看作“形式”公式。之所以称为“形式”公式，主要是因为在相关讨论中并不涉及这些公式的语义，仅将其看作一些符号的组合，而形式公式正是数理逻辑中语法系统研究的对象。通常需要按照上述 3 条公理，依据推理规则，由 F 中“形式”公式“逻辑推导”出新的“形式”公式。如果“形式”公式 Q 可通过 Armstrong 公理系统由“形式”公式 P“逻辑推导”得到，则记为 $P \vdash Q$。

（2）推理规则。以 Armstrong 基本公理 A_1、A_2 和 A_3 为基础，可以得出下面 5 条推理规则，当然，这些规则的“结论”公式也是“形式”公式。

【定理 2-1】 下述论断成立。

① A_4（合并性规则，union）：$\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$。

② A_5（分解性规则，decomposition）：$\{X \rightarrow Y, Z \subseteq Y\} \vdash X \rightarrow Z$。

③ A_6（拟传递性规则，pseudotransivity）：$\{X \rightarrow Y, WY \rightarrow Z\} \vdash WX \rightarrow Z$。

④ A_7（复合性规则，composition rule）：$\{X \rightarrow Y, W \rightarrow Z\} \vdash WX \rightarrow YZ$。

⑤ A_8（通用一致性规则，general unification rule）：$\{X \rightarrow Y, W \rightarrow Z\} \vdash X(W-Y) \rightarrow YZ$。

【例 2-21】 由合并性规则 A_4 和分解性规则 A_5，可以得到如下结论。

如果 $A_1A_2 \cdots A_n$ 是关系模式 $R(U)$ 的属性集，则 $X \rightarrow A_1A_2 \cdots A_n$ 的充分必要条件是 $X \rightarrow A_i$（$i=1,2,\cdots,n$）成立。

3. 有效性和完备性

如果由 F 出发根据 Armstrong 公理逻辑推导出的每一个“形式”公式 $X \rightarrow Y$ 作为“函数依赖”$X \rightarrow Y$ 都在 F^+ 当中，则称 Armstrong 公理系统是有效的。

如果 F^+ 中每个函数依赖 $X \rightarrow Y$ 都可以通过将 F 中元素作为“形式”公式，由此再根据 Armstrong 公理系统“逻辑推导”而得到，则称 Armstrong 公理系统是完备的。

公理系统的有效性保证了所有逻辑推导出的“形式”公式作为函数依赖都是语义为真，即逻辑推导出的形式公式都是函数依赖；公理系统的完备性保证了可以逻辑推导出所有可能的函数依赖，即所有函数依赖都可以由逻辑推导得到，或者说不能用公理系统逻辑推导的函数依赖都不能为真。

1）Armstrong 公理系统的有效性

【定理 2-2】 Armstrong 公理系统具有有效性。

证明：所有由 Armstrong 公理系统逻辑推导的所有公式的有效性取决于系统中 A_1、A_2 和 A_3 三公理的结论公式是否具有有效性。因此只需按照 F^+ 概念，证明当 3 条公理条件中

的“形式”公式属于 F 时，相应 3 条结论的“形式”公式属于 F^+ 即可。

(1) 自反律 A_1。因为在任何一个关系中不可能存在两个元组在属性 X 上的值相等而在 X 的某个子集 Y 上的值不相等，所以自反律结论公式属于 F^+。

(2) 增广律 A_2。反设如果关系模式 $R(U)$ 中某个关系 r 中存在两个元组 t 和 s 违反了 $XZ \to YZ$，即 $t[XZ]=s[XZ] \Rightarrow t[YZ] \neq s[YZ]$。

由 $t[XZ]=s[XZ] \Rightarrow t[X]=s[X]$ 和 $t[Z] \neq S[Z]$。由 $t[YZ] \neq s[YZ] \Rightarrow t[Y] \neq s[Y]$ 或 $t[Z] \neq s[Z]$。如果 $t[Y] \neq S[Y]$，结合 $t[X]=s[X]$，与 $X \to Y$ 成立矛盾，而 $t[Z] \neq s[Z]$ 不可能成立。这样就与增广律条件“$X \to Y$ 在 $R(U)$ 上成立”矛盾，所以增广律结论公式属于 F^+。

(3) 传递律 A_3。反设 $R(U)$ 的某个关系实例 r 中存在两个元组 t 和 s 违反了 $X \to Z$，即 $t[X]=s[X]$，但 $t[Z] \neq s[Z]$。而对于 $t[Y]$ 和 $s[Y]$ 来说，只能有下述两种情形。

① 如果 $t[Y] \neq s[Y]$，则与 $X \to Y$ 成立矛盾。

② 如果 $t[Y]=s[Y]$ 而 $t[Z] \neq s[Z]$，就与 $Y \to Z$ 成立矛盾。

无论哪种情况都导致矛盾，由此可知传递律的结论公式属于 F^+。

2) Armstrong 公理系统的完备性

为证明 Armstrong 公理系统的完备性，需引入基于初始函数依赖集合 F 的属性集合 X 的闭包概念。

属性闭包：设 F 是属性集合 U 上的一个函数依赖集，$X \subseteq U$，称

$$X_F^+ = \{A \mid A \in U, X \to A \text{ 由 } F \text{ 按照 Armstrong 公理系统推导得到}\}$$

为属性集 X 关于 F 的闭包。这里，可以将属性 A 看作 U 中的单属性子集。

如果问题讨论过程中只涉及一个确定函数依赖集 F，就无须对函数依赖集进行区分，属性集 X 关于 F 的闭包可简记为 X^+。需要注意的是，总有 $X \subseteq X^+ \subseteq U$。

例如，设有关系模式 $R(U,F)$，其中 $U=ABC$，$F=\{A \to B, B \to C\}$，按照属性集闭包概念，则有 $A^+=ABC$，$B^+=BC$，$C^+=C$。

【定理 2-3】 Armstrong 公理系统具有完备性质。

证明：只需证明“不能由 F 使用 Armstrong 公理系统推导的函数依赖不在 F^+ 中”。

设 F 是属性集合 U 上的一个函数依赖集合，并设 $X \to Y$ 不能从 F 通过 Armstrong 公理系统推导出来。需要证明，在题设之下，$X \to Y$ 不在 F^+ 当中，即至少存在一个关系 r 满足 F，但不满足 $X \to Y$。证明分为如下三步进行。

首先，具体构造 r。设 r 由两个元组 t_1 和 t_2 组成，其中，t_1 在 U 中全部属性上取值都为 1，t_2 在 X^+ 属性上取值为 1，而在其他属性上取值为 0，如表 2-3 所示。

表 2-3　关系实例 r 的构造

	X^+ 中属性值	$U \setminus X^+$ 中属性值
元组 t_1	11…1	11…1
元组 t_2	11…1	00…0

其次，证明 r 满足 F，即关系实例 r 满足 F 中所有函数依赖。

设 $X_0 \to Y_0$ 是 F 中任意一个函数依赖，分以下两种情况考虑。

① 如果 $X_0 \subseteq X^+$，则可知 $X \to X_0$。再由假设 $X_0 \to Y_0$，根据传递律 A_3，得到 $X \to Y_0$，从

而 $Y_0 \subseteq X(1)^+$。按照关系实例 r 定义，可知其在 X^+ 中属性值都相等。由 $X_0 \subseteq X^+$ 可知 $t_1[X_0]=t_2[X_0]$；由 $Y_0 \subseteq X^+$ 可知 $t_1[Y_0]=t_2[Y_0]$，于是 $X_0 \to Y_0$ 在 r 上成立。

② 如果 $X_0 \not\subseteq X^+$，即 X_0 中含有 X^+ 之外的属性。由关系实例 $r=\{t_1, t_2\}$ 定义，$t_1[X_0] \neq t_2[X_0]$。由数理逻辑关于蕴含式取真值的理论，$X_0 \to Y_0$ 自然成立。

由上述①和②可知，关系实例 r 满足 F 中的每个函数依赖。

最后，证明对于关系实例 r，$X \to Y$ 不成立。

由题设 $X \to Y$ 不能基于 F 通过 Armstrong 公理逻辑推导得出。按照 X^+ 定义，可以得到 $Y \not\subseteq X^+$。由关系实例 r 构造和 $X \subseteq X^+$ 可知 $t_1[X]=t_2[X]$；由 $Y \not\subseteq X^+$ 可知 $t_1[Y] \neq t_2[Y]$。由此，$X \to Y$ 在 r 上不成立。这样就证明，只要 $X \to Y$ 不能从 F 通过 Armstrong 公理逻辑推导得出，F 就不能逻辑蕴含 $X \to Y$。

2.3.3 关系模式范式

范式是符合某一种级别要求的关系模式集合。关系数据库中的关系必须满足一定的要求。满足不同程度的要求就形成了不同的范式。某一关系模式 R 为第 n 范式，可简记为 $R \in n\mathrm{NF}$。一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这个过程就称为规范化。

1）第一范式

如果一个关系模式 R 的所有属性都是不可分的基本数据项，则 $R \in 1\mathrm{NF}$。1NF 是对关系模式的关键要求。不满足 1NF 的数据库模式不能称为关系数据库。对于不满足 1NF 的关系模式，需要将不满足原子性要求的属性"分解"成为满足原子性的多个属性。

2）第二范式

满足 1NF 的关系模式并不一定是一个好的关系模式。

【例 2-22】 关系模式 S-L-C($\underline{\text{Sno,Cno}}$,Sdept,Sloc,Grade)，键属性为 Sno 和 Cno 组合属性。其中，Sloc 为学生住处，假设每个系的学生住在同一个地方。

函数依赖包括 $(\text{Sno, Cno}) \xrightarrow{P} \text{Grade}$，$\text{Sno} \to \text{Sdept}$，$(\text{Sno, Cno}) \xrightarrow{P} \text{Sdept}$，$\text{Sno} \to \text{Sloc}$，$(\text{Sno,Cno}) \xrightarrow{P} \text{Sloc}$，$\text{Sdept} \to \text{Sloc}$。

这里，非主属性 Sdept、Sloc 部分函数依赖于键，这将导致数据冗余，出现操作异常。解决方法是将 S-L-C 分解为 SC($\underline{\text{Sno,Cno}}$,Grade) 和 S-L($\underline{\text{Sno}}$,Sdept,Sloc) 两个关系模式，以消除这些部分函数依赖关系。

若 $R \in 1\mathrm{NF}$，且每一个非主属性都是完全函数依赖于键，则 $R \in 2\mathrm{NF}$。如果关系 R 的键只有单一一个属性，它就一定符合第二范式（前提是该关系模式符合第一范式）。

【例 2-23】

$$\text{S-L-C}(\underline{\text{Sno,Cno}},\text{Sdept,Sloc,Grade}) \in 1\mathrm{NF}$$
$$\text{S-L-C}(\underline{\text{Sno,Cno}},\text{Sdept,Sloc,Grade}) \notin 2\mathrm{NF}$$
$$\text{SC}(\underline{\text{Sno,Cno}},\text{Grade}) \in 2\mathrm{NF}$$
$$\text{S-L}(\underline{\text{Sno}},\text{Sdept,Sloc}) \in 2\mathrm{NF}$$

3）第三范式

关系模式 $R<U,F>$ 中若不存在这样的键 X、属性组 Y 及非主属性 $Z(Z \not\subseteq Y)$，使得

$X \rightarrow Y, Y \rightarrow Z$ 成立，则称 $R<U,F> \in$ 3NF。

$R \in$ 3NF，必有 $R \in$ 2NF。

当 $\boldsymbol{R} \in$ 3NF 时，则每一个非主属性既不部分依赖于键也不传递依赖于键。

【例 2-24】 2NF 关系模式 S_L(Sno,Sdept,Sloc)中有函数依赖 Sno→Sdept、Sdept→Sloc 和 Sno→Sloc。此时，Sno→Sloc 是 S_L 中非主属性对键的传递函数依赖，S_L 不属于 3NF。违反 3NF 的关系模式同样会导致数据冗余，出现操作异常。解决方法采用投影分解法，把 S_L 分解为 S_D(Sno,Sdept)和 D_L(Sdept,Sloc)两个关系模式，以消除传递函数依赖。此时，S_D 的键为 Sno，D_L 的键为 Sdept，两者都不再存在传递依赖。

4) BC 范式

关系模式 $R<U,F> \in$ 1NF，若 $X \rightarrow Y$ 且 $Y \not\subseteq X$ 时，X 必含有键，则 $R<U,F> \in$ BCNF。$R<U,F> \in$ BCNF等价于每一个决定因素属性组都包含主键。当 $R \in$ BCNF 时，所有非主属性对每一个键都是完全函数依赖；所有的主属性对每一个不包含它的键，也是完全函数依赖；没有任何属性完全函数依赖于非键的任何一组属性。

$R \in$ BCNF 必有 $R \in$ 3NF。

当 $R \in$ 3NF，且 R 只有一个候选键时，则 $R \in$ BCNF。

目前关系数据库有 6 级范式：第一范式(1NF)、第二范式(2NF)、第三范式(3NF)、巴斯-科德范式(BCNF)、第四范式(4NF)和第五范式(5NF，又称完美范式)。满足最低要求的范式是第一范式(1NF)。在第一范式的基础上进一步满足更多规范要求的称为第二范式(2NF)，其余范式依次类推。后面继续探讨 4NF 和 5NF 原理。

2.3.4 多值依赖与连接依赖

在关系模式中，数据之间存在一定的依赖或联系，对这种联系处理适当与否直接关系到模式中数据冗余情况。多值依赖和连接依赖是更为一般的数据依赖联系。

1. 多值依赖与 4NF

函数依赖实质上反映的是“多对一”联系，在实际应用中还会有“一对多”形式的数据联系，诸如此类不同于函数依赖的数据联系也会产生数据冗余，从而引发各种数据异常现象。

【例 2-25】 设有一个课程安排关系，如表 2-4 所示。

表 2-4 课程安排示意图

课程名称	任课教师	选用教材名称	课程名称	任课教师	选用教材名称
数学分析	T_{11}	B_{11}	数据结构	T_{21}	B_{21}
	T_{12}	B_{12}		T_{22}	B_{22}
	T_{13}			T_{23}	B_{23}

这里课程安排具有如下语义。

① “数学分析”这门课程可以由 3 个教师担任，同时有两本教材可以选用。

② “数据结构”这门课程可以由 3 个教师担任，同时有 3 本教材可供选用。

如果分别用 C_n、T_n 和 B_n 表示“课程名称”“任课教师”和“教材名称”，上述情形可以表示关系 CTB，如表 2-5 所示。

表 2-5　关系 CTB

C_n	T_n	B_n
数学分析	T_{11}	B_{11}
数学分析	T_{11}	B_{12}
数学分析	T_{12}	B_{11}
数学分析	T_{12}	B_{12}
数学分析	T_{13}	B_{11}
数学分析	T_{13}	B_{12}
数据结构	T_{21}	B_{21}
数据结构	T_{21}	B_{22}
数据结构	T_{21}	B_{23}
数据结构	T_{22}	B_{21}
数据结构	T_{22}	B_{22}
数据结构	T_{22}	B_{23}
数据结构	T_{23}	B_{21}
数据结构	T_{23}	B_{22}
数据结构	T_{23}	B_{23}

这个关系表是数据高度冗余的，通过分析可以发现它有如下特点。

① 属性集$\{C_n\}$与$\{T_n\}$间及$\{C_n\}$与$\{B_n\}$间存在数据依赖关系，而这两个数据依赖都不是"函数依赖"，因为当属性子集$\{C_n\}$的一个值确定之后，另一属性子集$\{T_n\}$就有一组值与之对应。例如，当 C_n 一个值"数学分析"确定后就有一组任课教师 T_n 值"T_{11}""T_{12}"和"T_{13}"与之对应。对于 C_n 与 B_n 也是如此。这是一种"一对多"的情形。

② 属性集$\{T_n\}$和$\{B_n\}$也有通过$\{C_n\}$建立起来的间接关系，这种关系值得注意的是，当$\{C_n\}$值确定后，所对应的一组$\{T_n\}$值与 $U-\{C_n\}-\{T_n\}$无关。例如，取定$\{C_n\}$值为"数学分析"，则对应$\{T_n\}$的一组值"T_{11}、T_{12}和 T_{13}"与此"数学分析"课程选用教材即 $U-\{C_n\}-\{T_n\}$值无关。这是"一对多"关系中的一种特殊情况。

如果属性子集 X 与 Y 之间依赖关系具有上述特征，就不能为函数依赖关系所包容，需要引入新的概念予以刻画与描述，这就是多值依赖。

1）多值依赖

设有关系模式 $R(U)$，X、Y 是属性集 U 中的两个子集，而 r 是 $R(U)$中任意给定一个关系实例。如果下述条件成立，则称 Y 多值依赖(Multivalued Dependency)于 X，记为 $X\rightarrow\rightarrow Y$。

① 对于 r 在 X 上的一个确定的值(元组)，都有 r 在 Y 中一组值与之对应。

② Y 的这组对应值与 r 在 $Z=U-X-Y$ 中的属性值无关。

如果 $X\rightarrow\rightarrow Y$，但 $Z=U-X-Y\neq\varnothing$，则称其为非平凡多值依赖，否则称为平凡多值依赖。平凡多值依赖常见情形是 $U=X\cup Y$，此时 $Z=\varnothing$。

需要指出的是，函数依赖反映对属性值的约束。例如，S＃C＃→G，如果在一个元组中，当学号 S＃和课程号 C＃的值确定之后，对应成绩 G 属性值一定唯一，不能是多个。例如，在例 2-25 中，由 $C_n\rightarrow\rightarrow T_n$ 可知，如果给定排课关系中有元组(数学分析，T_{11}，B_{11})(表 2-5 第一行)和(数学分析，T_{12}，B_{12})(表 2-5 第四行)，则一定也有元组(数学分析，T_{11}，B_{12})(表 2-5 第二行)和(数学分析，T_{12}，B_{11})(表 2-5 第三行)。

2）第四范式——4NF

对于 $R(U)$中的任意两个属性子集 X 和 Y，如果对于任意非平凡多值依赖 $X\rightarrow\rightarrow Y$，$X$ 都为超键，则称 $R(U)$满足第四范式，记为 $R(U)\in$4NF。由 4NF 定义可知：

① 由于 $R(U)$上的函数依赖 $X\rightarrow Y$ 可看作多值依赖 $X\rightarrow\rightarrow Y$，如果 $R(U)$属于第四范式，此时 X 就是超键，因此 $X\rightarrow Y$ 满足 BCNF，即 4NF 中所有的函数依赖都满足 BCNF。

② 如 $X\rightarrow\rightarrow Y$ 是非平凡多值依赖，在 4NF 中，X 就是超键，此时多值依赖就是函数依

赖，即 4NF 中可能的多值依赖或平凡多值依赖，或者名义上为多值依赖的函数依赖。

可以粗略地说，$R(U)$满足第四范式必满足 BC 范式。但反之则不成立，所以 BC 范式不一定是第四范式。

在例 2-25CTB(C_n，T_n，B_n)唯一候选键是{C_n，T_n，B_n}，且没有非主属性，当然就没有非主属性对候选键的部分函数依赖和传递函数依赖，所以 CTB 满足 BC 范式。但在多值依赖 $C_n \to\to T_n$ 和 $C_n \to\to B_n$ 中的"C_n"不是键，所以 CTB 不属于 4NF。对 CTB 进行分解，得到 CTB_1 和 CTB_2，如表 2-6 和表 2-7 所示。

表 2-6　关系 CTB_1

C_n	T_n
数学分析	T_{11}
数学分析	T_{12}
数学分析	T_{13}
数据结构	T_{21}
数据结构	T_{22}
数据结构	T_{23}

表 2-7　关系 CTB_2

C_n	B_n
数学分析	B_{11}
数学分析	B_{12}
数据结构	B_{21}
数据结构	B_{22}
数据结构	B_{23}

在 CTB_1 中，$C_n \to\to T_n$，不存在非平凡多值依赖，$CTB_1 \in 4NF$。同理 $CTB_2 \in 4NF$。

2. 连接依赖与 5NF

更为一般的数据依赖是连接依赖。就像引入多值依赖之后，函数依赖就成为多值依赖的特例，引入连接依赖概念之后，多值依赖就可以作为连接依赖的特例。

1）多值依赖的无损连接定义

可以由例 2-25 分析多值依赖问题。由关系模式 CTB 的属性集合 $U=\{C_n,T_n,B_n\}$上的一个划分{{C_n}，{T_n}，{B_n}}可以得到 U 上的一个覆盖{{C_n，T_n}，{C_n，B_n}}，如果记$CTB_1=\Pi_{\{C_n,T_n\}}(CRT)$，$CTB_2=\Pi_{\{C_n,B_n\}}(CRT)$，容易验证下式成立：

$$CTB = CTB_1 \triangleright\triangleleft CTB_2 = \Pi_{\{C_n,T_n\}}(CRT) \triangleright\triangleleft \Pi_{\{C_n,B_n\}}(CRT)$$

这说明由属性分解得到的模式分解具有"无损连接分解"性质。将 CTB 换为一般关系模式，将 CTB 的划分{{C_n}，{T_n}，{B_n}}换为一般属性集划分，就得到多值依赖的另一等价定义。

设有关系模式 $R(U)$，而 X、Y 和 Z 是属性集 U 的一个划分。如果对于 R 的每一个关系实例 r，都成立 $r=\Pi_{\{X,Y\}}(r) \triangleright\triangleleft \Pi_{\{X,Z\}}(r)$，则称多值依赖 $X \to\to Y$ 在 $R(U)$上成立。

上述定义可以看作多值依赖的无损连接分解定义，其意义在于可以进行推广，因为上述定义实际上做出了关系模式 $R(U,F)$的一种分解：$\rho=\{R_1(U_1,F_1),R_2(U_2,F_2)\}$，其中 $U_1=XY$，F_1 是 X 和 Y 之间数据依赖的集合，$U_2=XZ$，F_2 是 X 和 Z 之间数据依赖的集合。这里，$U_1 \cup U_2=\{X,Y\} \cup \{X,Z\}=U$，即{$X$，$Y$}和{$X$，$Z$}是 U 上的一个覆盖。而 Y 多值依赖于 Y 的充分必要条件就是由此得到的模式分解 ρ 是"无损连接分解"。这里模式分解集合只有两个元素，如果考虑多于两个元素，就得到"连接依赖"的概念。

2）连接依赖

设有关系模式 $R(U)$，$\{U_1,U_2,\cdots,U_n\}$是属性集合 U 的一个覆盖，关系模式集合 $\rho=\{R_1,R_2,\cdots,R_n\}$是 R 的一个模式分解，其中 R_i 是对应于 U_i 的关系模式($i=1,2,\cdots,n$)。如果对于 R 的每一个关系实例 r，下式成立：

$$r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r) \bowtie \cdots \bowtie \prod_{R_n}(r)$$

则称连接依赖(Join Dependency,JD)在关系模式 R 上成立,记为 $\bowtie(R_1, R_2, \cdots, R_n)$。

如果连接依赖中每一个 $R_i, i=1,2,\cdots,n$ 都不等于 R,则称此时连接依赖是非平凡的,否则称为平凡的。

由连接依赖定义,多值依赖是模式的无损分解集合中只有两个分解元素的连接依赖,因而是连接依赖特例,连接依赖是多值依赖的推广。

【例 2-26】 供应关系 SPJ{S#,P#,J#},其中 S#,P# 和 J# 分别表示供应商编号、零件编号和工程编号。SPJ 表示供应关系,即某个供应商提供某零件给某工程。令 SP={S#,P#}、JP={P#,J#}和 JS={J#,S#},则存在连接依赖 $\bowtie$(SP,PJ,JS)在 SPJ 上成立。

设关系实例 $r_1=(S_0, P_0)\in$ SP,表示公司 S_0 供应零件 P_0。

设关系实例 $r_2=(J_0, P_0)\in$ JP,表示工程 J_0 需要零件 P_0。

设关系实例 $r_3=(S_0, J_0)\in$ SJ,表示公司 S_0 和 J_0 有供应零件关系。

此时,关系实例 $r=(S_0, P_0, J_0)=\Pi_{SP}(r)\bowtie\Pi_{JP}(r)\bowtie\Pi_{SJ}(r)=r_1\bowtie r_2\bowtie r_3\in$ SPJ 就表示公司 S_0 必须为工程 J_0 提供零件 P_0。

3) 第五范式——5NF

假设关系模式 $R(U)$ 上任意一个非平凡连接依赖 $\bowtie(R_1, R_2, \cdots, R_n)$ 都由 R 的某个候选键所蕴含,则称关系模式 R 满足第五范式,记为 $R(U)\in$ 5NF。

第五范式在有些文献中也称为投影连接范式(Project-Join Normal Form),简记为 PJNF。

这里所说的由 R 的候选键所蕴含,是指 $\bowtie(R_1, R_2, \cdots, R_n)$ 可以由候选键推出。

【例 2-27】 在例 2-26 $\bowtie$(SP,PJ,JS)中的 SP、PJ 和 JS 都不等于 SPJ,是非平凡的连接依赖,但 $\bowtie$(SP,PJ,JS)并不被 SPJ 的唯一候选键{S#,P#,J#}蕴含,因此不是 5NF。将 SPJ 分解成 SP、PJ 和 JS 3 个模式,此时分解是无损分解,并且每一个模式都是 5NF,可以消除冗余及其操作异常现象。

关系模式分解需要按照一定方式来保证原有信息不至于"畸变"或"损失"。分解实际上可以看作属性集合 U 的投影,而各种必要信息的保持主要是通过"连接"实现的。因此,从直观上来看,迄今为止采取的模式设计方法就是"使用投影进行分解"和"使用连接进行重构"。从这个角度来看,连接依赖就是所有基于"投影分解和连接重构"方法的最一般形式,同时 5NF 也就覆盖了所有以投影、连接为基础的各种规范化形式。当然,如果不限于上述分解方式,就有可能考虑其他的规范化方法。

3 种数据依赖和各种范式之间的关系如图 2-1 所示。

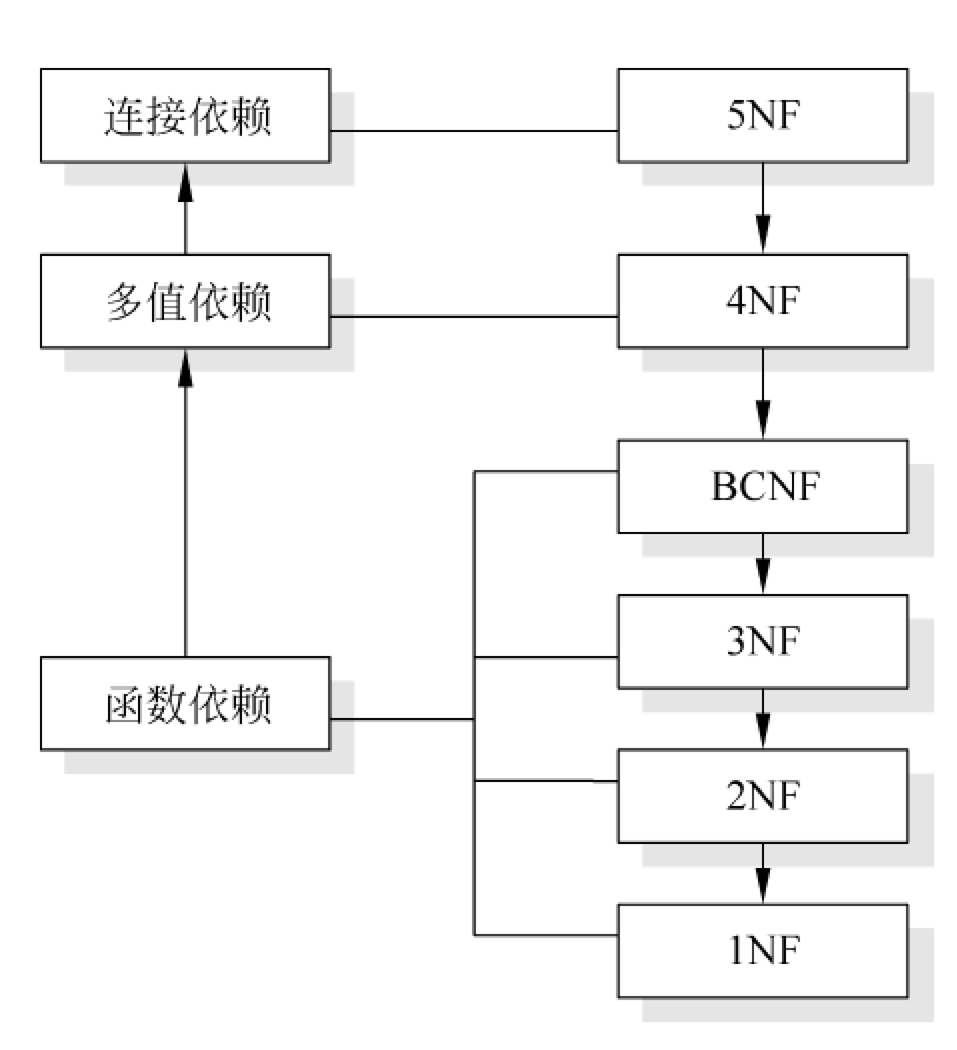


图 2-1　3 种数据依赖及各种范式之间的关系

2.4 关系数据库保护

关系数据库保护主要是指数据库的完整性保护和安全性保护。

完整性保护：防止数据库中存在不符合语义的数据，即防止数据库中存在不正确的数据，防范对象为不合语义的、不正确的数据。

安全性保护：防止恶意的破坏和非法的存取；防范对象为非法用户和非法操作。

以下简要回顾数据库保护技术要点及其在SQL中的实现。

2.4.1 完整性保护

完整性包括数据的正确性和相容性，主要是指数据的实体完整性、域完整性、参照完整性和用户定义完整性。

1）实体完整性保护

关系模型的实体完整性在基本表创建语句CREATE TABLE中用PRIMARY KEY定义。若主键由单属性构成，则可将其定义为列级约束条件；否则需将其定义为表级约束条件。

【例2-28】 定义S中的实体完整性。

```
CREATE TABLE S
   (Sno    CHAR(9) PRIMARY KEY,
    Sname  VARCHAR(20) NOT NULL
);
```

当插入数据或对主键列进行更新操作时，RDBMS按照实体完整性规则自动进行检查。首先检查主键值是否唯一，如果不唯一则拒绝插入或修改；其次检查主键的各个属性是否为空，只要有一个为空就拒绝插入或修改。

2）域完整性保护

域完整性保护是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。

【例2-29】 定义S中的域完整性。

```
CREATE TABLE S
   (Sno    CHAR(9) PRIMARY KEY,
    Sname  VARCHAR(20) NOT NULL,
    Sdate  DATATIME
);
```

对于Sdate列定义了数据类型为DATATIME类型，那么在进行数据输入或更新时，只能输入符合日期型数据类型的值，如'2017-01-01'，如果是输入'abcd'等普通字符串，RDBMS会进行检查并返回错误提示。

3）参照完整性保护

关系模型的参照完整性在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外键，用REFERENCES短语指明这些外键参照哪些表的主键。

例如，关系 SC 中一个元组表示一个学生选修的某门课程的成绩，(Sno，Cno)是主键。Sno，Cno 分别参照引用 Student 表的主键和 Course 表的主键。

【例 2-30】 定义 SC 中的参照完整性。

```
CREATE TABLE SC
   (Sno    CHAR(9) NOT NULL,
    Cno    CHAR(4) NOT NULL,
    Grade  SMALLINT,
    PRIMARY KEY (Sno, Cno), /* 此处为多属性主键,必须为表级约束条件 */
    FOREIGN KEY (Sno) REFERENCES Student(Sno),
    FOREIGN KEY (Cno) REFERENCES Course(Cno)
    );
```

在可能出现破坏参照完整性情形时 DBMS 会进行相应处理，如表 2-8 所示。

表 2-8 违反参照完整性处理

被参照表(如 S 表)	参照表(如 SC 表)	违反参照完整性处理
可能破坏参照完整性	←插入元组	拒绝
可能破坏参照完整性	←修改外键值	拒绝
删除元组→	可能破坏参照完整性	拒绝/级联删除/设置为空值
修改主键值→	可能破坏参照完整性	拒绝/级联修改/设置为空值

如表 2-8 所示，违反参照完整性有 3 种处理方式：拒绝(NO ACTION)执行，这是系统默认策略；级联(CASCADE)删除或设置为空值(SET-NULL)。

对于参照完整性，除了应该定义外键外，还应定义外键列是否允许空值，默认为可以为空。

4) 用户定义完整性保护

按照定义方式的不同，用户定义完整性约束分为创建关系时定义和独立定义两种类型。

(1) 创建关系时定义。此时有属性约束条件和元组约束条件两种情形。

① 属性列完整性约束。在 CREATE TABLE 时定义，主要有属性值非空(NOT NULL)约束、属性值唯一(UNIQUE)约束和属性值是否满足一个布尔表达式(CHECK)约束。

【例 2-31】 Student 表的 Ssex 只允许取“男”或“女”。

```
CREATE TABLE Student
    (Sno   CHAR(9) PRIMARY KEY,
     Sname CHAR(8) NOT NULL,
     Ssex  CHAR(2) CHECK (Ssex IN ('男','女') ),
     Sage  SMALLINT,
     Sdept CHAR(20)
     );
```

说明： *这是使用 CHECK 短语定义属性上约束的情形，可以作为列级约束条件。*

当插入元组或修改属性值时，RDBMS 会自动检查属性上的约束条件是否被满足，如果不满足则操作被拒绝执行。

② 元组完整性约束。在 CREATE TABLE 时可以用 CHECK 短语定义元组上的约束

条件，即元组级的限制。与属性值限制相比，元组级限制可设置不同属性间取值的相互约束条件。

【例 2-32】 学生的性别是男时，其名字不能以 Ms. 打头。

```
CREATE TABLE Student
      (Sno    CHAR(9),
       Sname  CHAR(8) NOT NULL,
       Ssex   CHAR(2),
       Sage   SMALLINT,
       Sdept  CHAR(20),
       PRIMARY KEY (Sno),
       CHECK (Ssex = '女' OR (Ssex = '男' AND Sname NOT LIKE 'Ms. % '))
       );
```

性别是女性的元组都能通过该项检查，因为 Ssex='女'成立；当性别是男性时，同时名字需要通过检查一定不能以 Ms. 打头，由于 Check 涉及表中的多个属性，只能使用表级约束条件。

插入元组或修改属性的值时，RDBMS 检查元组上的约束条件是否被满足，如果不满足则操作被拒绝执行。

(2) 独立定义用户完整性约束。传统的独立定义用户完整性约束分为断言约束、规则约束和域约束 3 种情形。

由于目前流行的关系数据库产品出于性能的考虑，基本不支持这些约束，而会使用触发器等编程功能来实现。

2.4.2 安全性保护

数据库安全性涉及技术安全、管理安全和政策安全 3 个方面，在数据库学科中主要讨论数据库的技术安全，其中包括操作系统安全、网络安全和数据库自身安全技术。通常数据库课程中讨论的数据库安全保护技术主要是指数据库自身安全技术。

数据库自身安全技术的基础是数据库安全性控制、视图和审计，其中安全性控制是数据库安全性保护的主要技术支撑。

1) 基于存取控制的安全性技术

数据库安全性控制分为自主存取控制和强制存取控制两种情形。

(1) 自主存取控制。用户对不同的数据库对象具有不同的存取权限，同时，不同用户对同一数据库对象也有不同的存取权限。只有具有相应权限的用户才能存取相应的数据库对象。

用户权限由数据库对象和操作类型组成。数据库对象有数据库、基本表、视图和索引等；数据操作类型有创建、更新、查询等。

所有的关系数据库管理系统都支持自主存取控制。在 SQL 中，主要是通过 GRANT 和 REVOKE 语句实现自主存取控制。

【例 2-33】 把对 Student 表和 Course 表的全部权限授予用户 U2 和 U3。

```
GRANT ALL PRIVILIGES
ON TABLE Student,Course
TO U2,U3;
```

把对 SC 表的 INSERT 权限授予 U5 用户，并允许他再将此权限授予其他用户。

```
GRANT INSERT
ON TABLE SC
   TO U5
   WITH GRANT OPTION;
```

说明：WITH GRANT OPTION 短语标识 DBA 允许被授权用户可以将其得到的权限传递给其认为合适的其他用户，这将提高授权机制的效率。

【例 2-34】 将用户 U5 对 SC 表的 INSERT 权限收回。

```
REVOKE INSERT
ON TABLE SC
   FROM U5 CASCADE;
```

说明：如果被收回权限的用户还具有“传播”的权限，需要如同上述语句那样，在末尾添加 CASCADE。上述语句就表示将用户 U5 的 INSERT 权限收回时必须级联收回其传递给其他用户的 INSERT 权限，如果此时没有关键字 CASCADE，而 U5 又具有传递权限，则系统将拒绝执行收回语句。

（2）强制存取控制。每个数据库对象都分配一个“密级”，每个用户也被授予一个级别。对于每个数据库对象，根据其密级，只有具有相应合法许可证级别的用户才能存取。与自主存取控制相比，强制存取控制安全性检查更为严格。

在实际应用中，通常是在实现 MAC 之前先实现 DAC，从而由两者共同构成数据库的安全性保护。

2）基于视图的安全性保护

为不同用户创建不同视图，通过将数据对象限制在一定范围内，把需要保密的数据对无权存取的用户屏蔽起来，从而自动为数据提供一定程度的安全性保护。

3）基于审计的安全性保护

基于存取控制和基于视图的安全性机制可以看作是“事先预防”，但任何安全性措施都会有缺陷，都有可能被攻破。审计(audit)就是将用户对于数据库对象的所有操作通过审计日志予以记录，然后通过审计跟踪相应行迹，以便重现可能导致破坏数据库安全性的一系列事件，从而发现非法存取的人、事件和内容等。审计通常需要较大时间与空间开销，一般都是作为安全性的可选项目，即便是使用审计，也多针对个别情形，而不是整个数据库对象。审计功能适用于对安全性要求较高的环境。SQL 中通过 AUDIT 语句和 NOAUDIT 语句创建或撤销对给定数据库对象的审计功能。

【例 2-35】 对修改 SC 表结构或修改 SC 表数据的操作进行审计。

```
AUDIT ALTER,UPDATE
ON SC;
```

【例 2-36】 取消对 SC 表的一切审计。

```
NOAUDIT ALTER,UPDATE
ON SC;
```

2.5 关系数据库事务处理

利用数据库完成一项业务工作时常常会涉及多个数据库操作组成的序列，在技术上，每个操作都可独立完成，但在逻辑上，只有序列中每个操作都完成之后才能实现一项具体的业务工作。也就是说，这些可以单个执行的操作是有内在关联的，它们组成的操作序列需要看作一个整体，是一个不可分割的工作单元。事实上，现代关系数据库就是以如此的单元为其基本处理单位，这种工作单元就是数据库中的事务。

事务(transaction)是一个满足下述性质的数据库操作的序列。

(1) 工作执行的原子性(atomic)。序列中的数据操作“要么全做，要么全不做”，不能存在部分完成的情况。

(2) 更新操作的一致性(comsistency)。在执行数据更新过程中要保证数据前后的一致性，需要从一种一致性状态转换到另一种一致性状态。

(3) 并发执行的隔离性(isolation)。如果多个操作序列同时执行，其最终效果需要与单个操作序列独立执行一样。

(4) 成功结果的持久性(durability)。当操作序列成功执行后相应数据结果对于数据库影响持久，即便是当数据库发生故障而遭到破坏时也能恢复原先的数据结果。

也就是说，只有满足上述 ACID 性质的操作序列才是数据库意义下的事务，而事务是 DBMS 进行数据管理的执行工作单元。

如前所述，数据库的基本目标是实现对数据的统一管理和用户共享。

实现用户共享数据的技术途径就是多项数据库操作的并发执行，而进行统一管理数据的一项重要任务就是数据库的故障恢复，而事务管理就为“并发执行”和“故障恢复”提供了适当的和有效的技术实现架构。

2.5.1 并发控制

数据库的特性之一是数据共享，即多个用户同时使用数据库中同一数据，这就需要考虑数据库中多个事务的并发执行。多事务并发执行有以下 3 种实现方式。

(1) 事务串行执行。每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行，这种方式不能充分利用系统资源，发挥数据库共享资源的特点。

(2) 事务交叉并发方式(interleaved concurrency)。在单处理机系统中，事务的并行执行是这些并行事务的并行操作轮流交叉运行，单处理机系统中的并行事务并没有真正并行运行，但能够减少处理机的空闲时间，提高系统的效率。

(3) 事务同时并发方式(simultaneous concurrency)。多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行。

通常数据库多事务并发执行主要研究交叉并发执行。这里主要需要研究下述 3 个问题。

① 如果不进行并发控制，会出现哪些问题，又如何解决？

② 并发调度有多重方式，如何判定哪种方式是“合适”的？

③ 由于并发执行的复杂性，在上述两个问题之外，还会产生什么问题？如何解决？

1. 并发控制封锁技术

1）并发控制的必要性

并发执行需要加以控制或调度，否则会出现下述问题。

（1）丢失修改。例如，在飞机售票系统中，不同售票点可能同时发售机票。

① 甲售票点（甲事务）读出某航班的机票余额 A，设 $A=16$。

② 乙售票点（乙事务）读出同一航班的机票余额 A，也为 16。

③ 甲售票点卖出一张机票，修改余额 $A \leftarrow A-1$，所以 A 为 15，把 A 写回数据库。

④ 乙售票点也卖出一张机票，修改余额 $A \leftarrow A-1$，所以 A 为 15，把 A 写回数据库。

结果明明卖出两张机票，数据库中机票余额只减少 1。此时，两个事务 T_1 和 T_2 读入同一数据并修改，T_2 的提交结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失（lost update）。

（2）不可重复读。不可重复读（non-repeatable read）是指事务 T_1 读取数据后，事务 T_2 执行更新操作，使 T_1 无法再现前一次读取结果。

不可重复读包括以下 3 种情况。

① 事务 T_1 读取某一数据后，事务 T_2 对其做了修改，当事务 T_1 再次读该数据时，得到与前一次不同的值，如图 2-2 所示。

在图 2-2 中，T_1 读取 $B=100$ 进行运算；T_2 读取同一数据 B，对其进行修改后将 $B=200$ 写回数据库；T_1 为了对读取值校对重读 B，B 已为 200，与第一次读取值不一致。

② 事务 T_1 按一定条件从数据库中读取了某些数据记录后，事务 T_2 删除了其中部分记录，当 T_1 再次按相同条件读取数据时，发现某些记录消失了。

③ 事务 T_1 按一定条件从数据库中读取某些数据记录后，事务 T_2 插入了一些记录，当 T_1 再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为幻影现象（phantom row）。

（3）读“脏”数据。“脏”数据（dirty read）是指事务 T_1 修改某一数据，并将其写回磁盘；事务 T_2 读取同一数据后，T_1 由于某种原因被撤销；这时 T_1 已修改过的数据恢复原值，T_2 读到的数据就与数据库中的数据不一致；T_2 读到的数据就为“脏”数据，即不正确的数据，如图 2-3 所示。

T_1	T_2
① $R(A)=50$	
$R(B)=100$	
$A+B=150$	
②	$R(B)=100$
	$B \leftarrow B*2$
	$W(B)=200$
③ $R(A)=50$	
$R(B)=200$	
$A+B=250$	
（校验出错）	

图 2-2　不可重复读

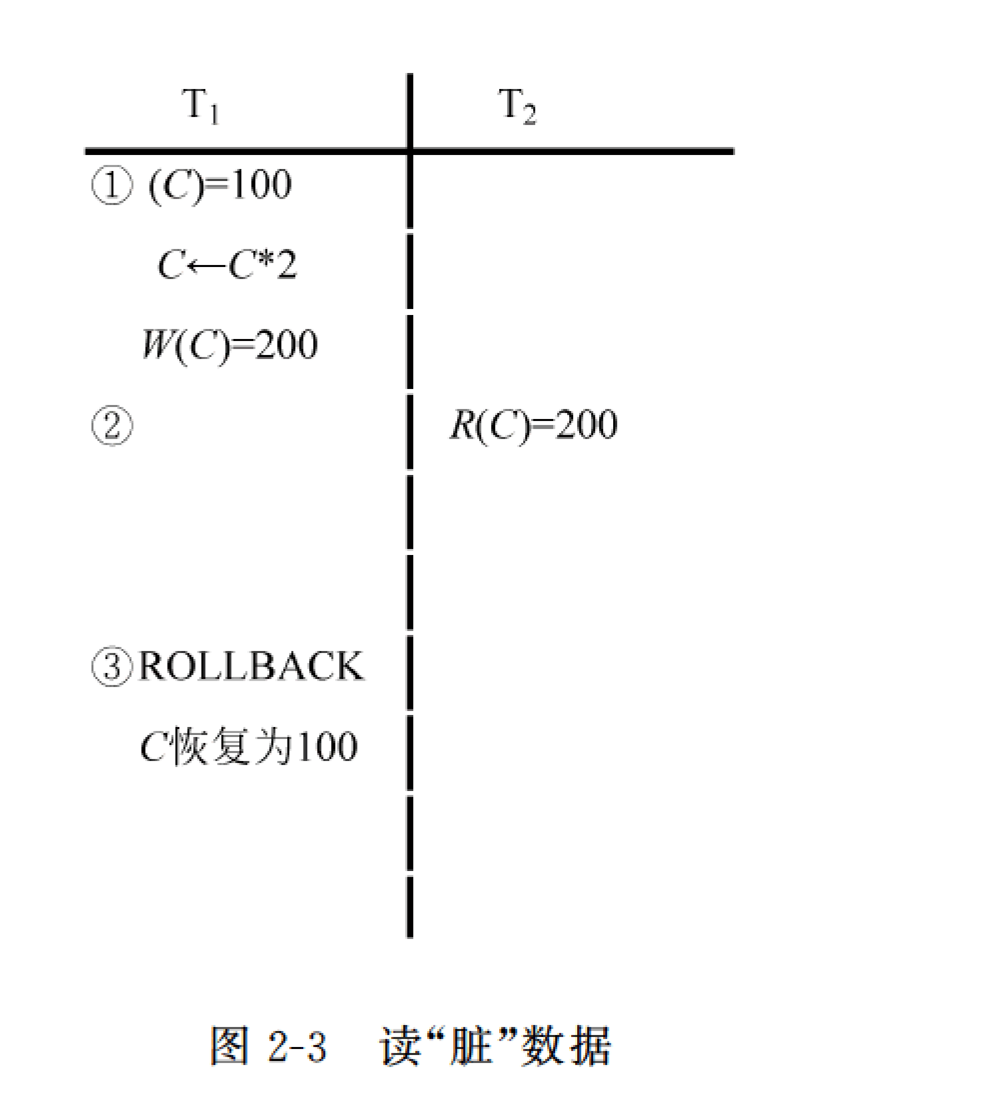


图 2-3　读“脏”数据

在图 2-3 中，T_1 将 C 值修改为 200，T_2 读到 C 值为 200，T_1 由于某种原因撤销，其修改作废，C 恢复原值 100；这时 T_2 读到的 C 值为 200，与数据库内容不一致，就是“脏”数据。

上述 3 种情形都会导致数据不一致性，这是由于不加控制的并发操作破坏了事务的隔离性。多事务并发执行需要进行适当的并发控制，而并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

2）封锁技术

并发控制技术就是事务的封锁技术。封锁是系统对事务并发执行的一种调度和控制技术，是保证系统对数据项的访问以互斥方式进行的一种手段。封锁技术的基本点在于对数据对象操作实行某种专有控制。在一段时间内，防止其他事务访问指定资源，禁止某些用户对数据对象做某些操作以避免不一致性，保证并发执行的事务之间相互隔离、互不干扰，从而保障并发事务的正确执行。

（1）当一个事务 T 需要对数据对象 D 进行操作（读/写）时，必须向系统提出申请，对 D 加以封锁；在获得加锁成功之后，即具有对数据 D 一定操作权限与控制权限，此时，其他事务不能对加锁的数据 D 随意操作。

（2）当事务 T 操作完成之后即释放锁，此后数据即可为其他事务操作服务。

基于封锁技术的事务进程如图 2-4 所示。

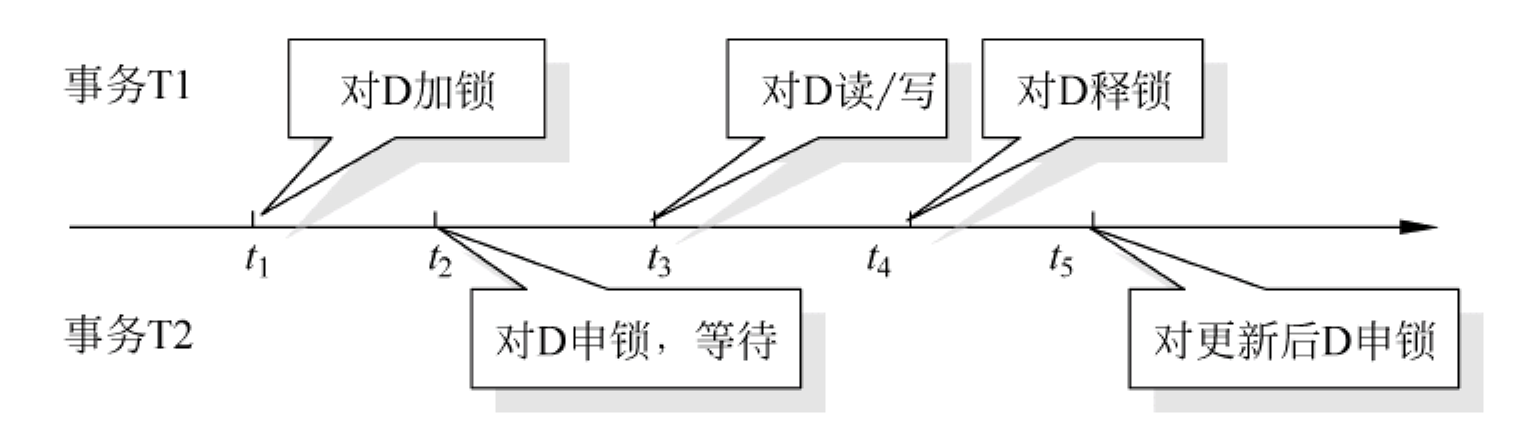


图 2-4　基于封锁技术的事务的进程

表级封锁可分为排他锁和共享锁两种形式。

① 排他锁（exclusive locks）：又称为写锁或 X 锁。其含义是事务 T 对数据对象 D 加 X 锁后，T 可以对加 X 锁的 D 进行读/写，而其他事务只有等到 T 解除 X 锁之后，才能对 D 进行封锁和操作（包括读/写）。

② 共享锁（sharing locks）：又称为读锁或 S 锁。其含义是事务 T 对数据 D 加 S 锁之后，T 可以读 D 但不能写 D；同时其他事务可以对 D 加 S 锁但不能加 X 锁。

X 锁和 S 锁如图 2-5 所示。

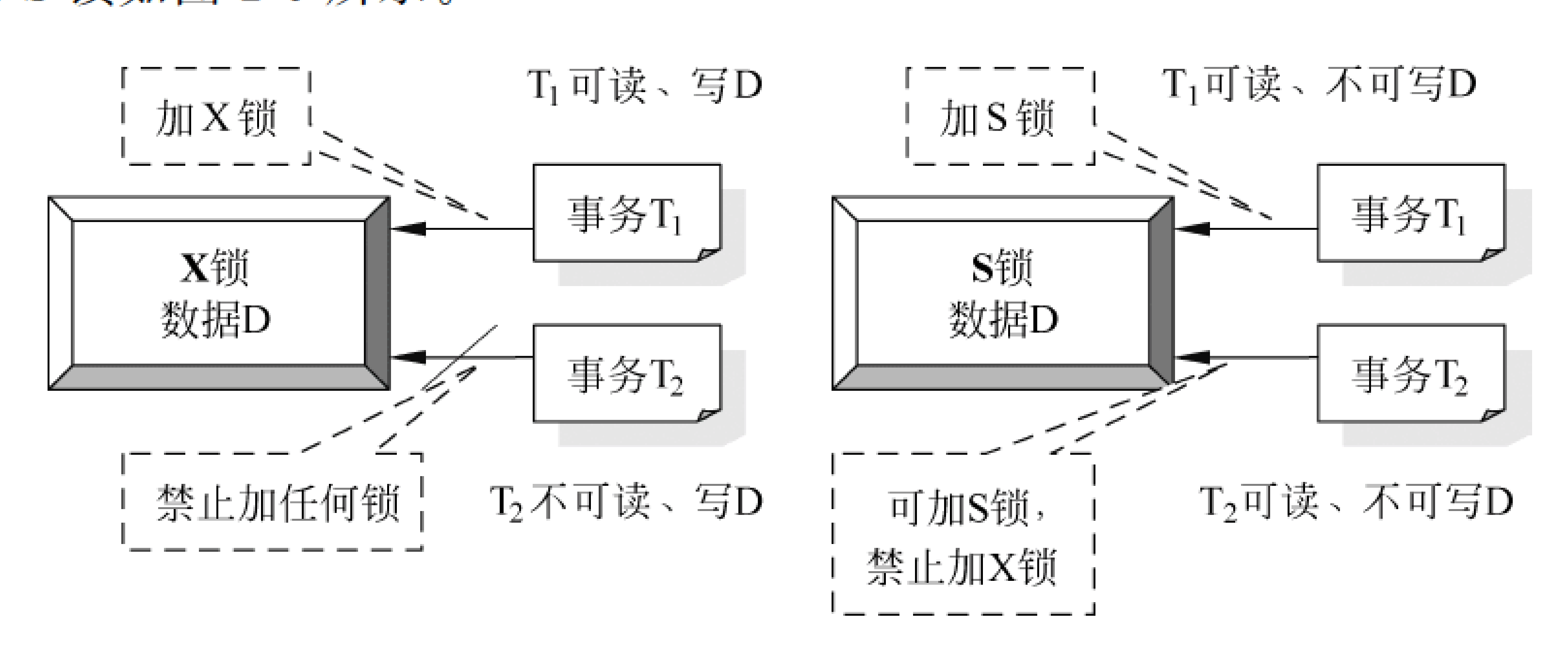


图 2-5　X 锁与 S 锁

排他锁和共享锁的控制方式也可用相容矩阵表示，如图 2-6 所示。

3）三级封锁协议

T_1 \ T_2	X	S	no lock
S	×	√	√
X	×	×	√

图 2-6　S 锁和 X 锁的相容矩阵

在对数据进行封锁时需要约定一些规则，如何时申请封锁、持锁时间和什么时间释放封锁等，这就是封锁协议（locking protocol）。

(1) 一级封锁协议。事务 T 在对数据 D 进行写操作之前，必须对 D 加 X 锁；保持加锁状态直到事务结束（包括 COMMIT 与 ROLLBACK）才可释放加在 D 上的 X 锁。由此防止了“丢失修改”。

(2) 二级封锁协议。事务 T 在读取数据 D 之前必须先对 D 加 S 锁，在读完之后即可释放加在 D 上的 S 锁。此封锁方式与一级封锁协议一起构成二级封锁协议。由此防止了“读脏数据”。

(3) 三级封锁协议。事务 T 在对数据 D 读之前必须先对 D 加 S 锁，直到事务结束才能释放加在 D 上的 S 锁。这种封锁方式与一级封锁协议一起就构成三级封锁协议。由于包含一级封锁协议，防止了丢失修改；包含二级封锁协议的基本内容，防止了读脏数据；另外由于在对数据 D 做“写”操作时加 X 锁封锁，做“读”操作时加 S 锁封锁，这两种锁都直到事务结束后才释放，由此防止不可重复读。三级封锁协议同时防止并发执行中的 3 类问题。

2. 可串行化调度

1）串行化调度

数据库中多事务并发执行是基于各个事务中操作的“交错”执行，其中每一种交错方式就称为并发执行的一种调度方式。并发执行的调度方式非常之多，DBMS 对并发事务不同的调度可能会产生不同的结果，因此，需要明确哪一种方式是“正确”的，哪一种方式是“不正确”的。从直观上来讲，多个事务的“顺序”即“串行”执行结果总是不会出错的，因此可以多事务的串行执行作为并发调度的正确性标准。

可串行化（serializable）调度：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。

2）两段封锁协议

两段封锁协议（Two-Phase Locking，2PL）主要是要求所有事务必须分两个阶段对数据项加锁和解锁，从而保证了由其产生的并发调度是可串行化的。

在 2PL 中，事务的封锁分为以下两个阶段。

阶段 1：获得封锁，也称为扩展阶段，此时，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁。

阶段 2：释放封锁，也称为收缩阶段，此时，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁。

当事务 T 遵守 2PL 时，相应封锁序列过程如下：

Slock A　Slock B　Xlock C　Unlock B　Unlock A　Unlock C

|←　　扩展阶段　　→|　|←　　收缩阶段　　→|

当事务不遵守两段锁协议，相应封锁序列过程如下：

Slock A　Unlock A　Slock B　Xlock C　Unlock C　Unlock B

2.5.2 故障恢复

数据库的恢复是把数据库从错误状态恢复到某一已知的正确状态(也称为一致状态或完整状态)。

1. 数据库故障类型

数据库的故障主要有事务级(事务内部)故障、系统级故障和介质级故障3种类型。

(1) 事务级故障。事务内部故障多是非预期的,不能由应用程序处理,如运算溢出、并发事务发生死锁而被选中撤销该事务和违反了某些完整性限制等。事务故障恢复策略主要是撤消事务(UNDO)。

(2) 系统级故障。系统级故障也称为软故障,是指造成系统停止运转的任何事件,使得系统需要重新启动,如整个系统的正常运行突然被破坏、所有正在运行的事务都非正常终止等。其特征是不破坏数据库但内存中数据库缓冲区的信息全部丢失。

(3) 介质级故障。介质级故障也称为硬故障,主要是指外存故障,如磁盘损坏、磁头碰撞、操作系统的某种潜在错误和瞬时强磁场干扰等。介质级故障恢复策略主要是装入数据库发生介质故障前某个时刻数据副本并重做自此时所有成功事务,将这些事务已提交的结果重新记入数据库。

2. 故障恢复技术

数据故障恢复技术就是冗余技术,即利用存储在系统其他位置的冗余数据来重建数据库中已被破坏或不正确的那部分数据。恢复机制涉及的关键问题如下。

① 建立冗余数据方式:主要技术为数据备份(backup)和登录日志文件(logging)。

② 使用冗余数据策略:利用冗余数据实施数据库恢复。

1) 数据备份与日志

数据备份是指DBA将整个数据库复制到磁带或另一个磁盘上保存起来的过程,备用的数据称为后备副本或后援副本。数据库遭到破坏后可以将后备副本重新装入但重装后备副本只能将数据库恢复到转储时的状态。

日志文件(log)是用来记录事务对数据库的更新操作的文件,主要分为以记录为单位和以数据块为单位的两类日志文件。

2) 基于备份和日志的故障恢复

基于备份与日志的数据库故障恢复原理如图2-7所示。

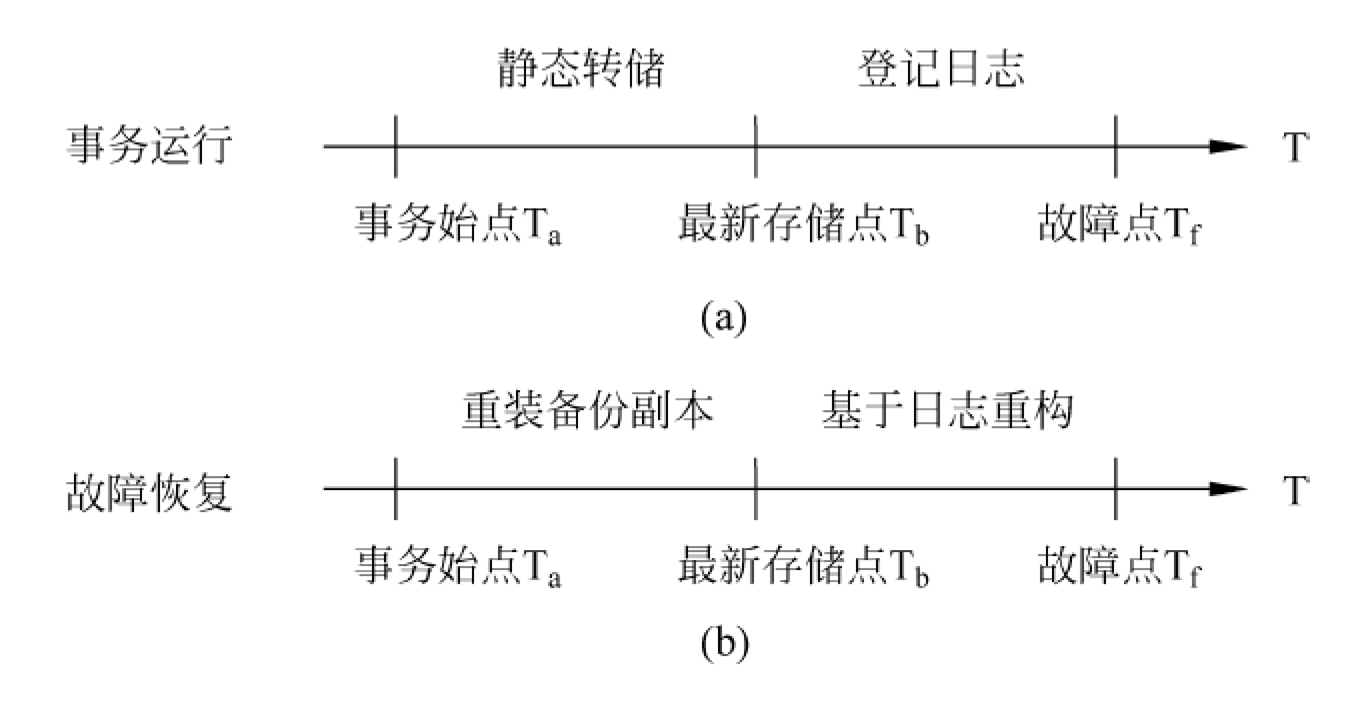


图2-7 数据库故障恢复原理

在图 2-7 中，系统在 T_a 时刻停止运行事务，进行数据库转储；在 T_b 时刻转储完毕，得到 T_b 时刻的数据库一致性副本；系统运行到 T_f 时刻发生故障；为恢复数据库，首先由 DBA 重装数据库后备副本，将数据库恢复至 T_b 时刻的状态；重新运行自 T_b～T_f 时刻的所有更新事务，把数据库恢复到故障发生前的一致状态。

（1）事务级故障恢复。当事务在运行至正常终止点前被终止时，由恢复子系统应利用日志文件撤销（UNDO）此事务已对数据库进行的修改。事务故障恢复由系统自动完成，对用户透明，不需要用户干预。

（2）系统级故障恢复。此时造成数据库不一致状态的原因主要是未完成事务对数据库的更新已写入数据库；已提交事务对数据库的更新还留在缓冲区没来得及写入数据库。此时，可通过 UNDO 故障发生时未完成的事务，通过 REDO 已完成的事务。系统故障的恢复由系统在重新启动时自动完成，不需要用户干预.

（3）介质级故障恢复。需要 DBA 介入，其主要工作是重装最近转储的数据库副本和有关的各日志文件副本和执行系统提供的恢复命令，但具体的恢复操作仍由 DBMS 完成。

本章小结

关系数据库技术是迄今为止研究最为完整和开发最为成熟的数据管理技术，其核心是关系数据模型。关系数据模型从数学原理上看是有限个域上笛卡儿乘积的一个子集（这种子集在数学中称为“关系”，这也是“关系数据库”中“关系”一词的由来），从技术上看是进行必要限制的规范化了的二维“平面表格”（因此关系数据文件也称为“关系表”）。关系数据模型在具体应用层面的实现就是“关系模式”，具体而言就是根据实际要求设计的关系表格；填写了实际数据后就成为“关系实例”。关系数据库只有一个统一的关系数据模型，但对应一个实际应用，就会有一个或多个数据关系模式；一个关系模式通过不同的外延界定就可以对应众多的关系实例。例如，所有关系型数据库（Oracle、DB2 和 SQL Server 等）本质上都是基于同一个关系数据模型；给定其中一个系统平台，建立一个具体的计算机学院数据库就需要定义“教师”“学生”“教学”和“科研”等关系模式，而模式“教师”可以对应“软件工程系”“网络工程系”等具体关系实例，对于“学生”模式也对应“2015 级”“2016 级”和“2017 级”等关系实例。

关系数据库以关系模型为核心，在关系模型基础上创建各类关系模式，导入基于各类关系模式的实际应用关系实例（关系表），从而实现了关系数据的有效管理。在这其中，系统级别上的关系模式管理机制尤为重要，这就是“三级模式/两级映射”的关系数据库体系结构。作为一个管理系统首先需要建立其体系结构，体系结构设计需要考虑其应用特性。数据库最重要的应用特性是“共享性”，为保障共享性的正确及有效实施，就需要将数据库的用户视图层面、逻辑层面和物理存储层面进行适当“隔离”，也就是需要构建数据的“独立性”。用户层与逻辑层的“隔离”称为“逻辑独立性”；用户层和存储层的“隔离”称为“物理独立性”。关系数据库通过“外模式”（子模式）、模式（逻辑模式）和内模式（物理模式）建立起“三级模式”，通过“外模式/模式映射”和“模式/内模式映射”建立起“两级映射”。这种“三级模式/两级映射”就构成了关系数据库的体系结构，其中“外模式/模式映射”保障关系数据的逻辑独立性；“模式/内模式映射”保障物理独立性。现有的关系数据库都具有这种体系结构，其意义可以

类比计算机中的冯・诺依曼体系结构。

关系数据库的核心——关系数据模型建立在数学基础之上，基于关系数据模型的数据操作也就能够使用相应的数学运算进行描述，这就是关系运算。关系运算是关系数据库语言的理论基础，其中包括“关系代数”和“关系演算”，前者可以看作是数学中“集合论”的应用与扩展，后者是建立在“数理逻辑”基础之上的。只有掌握关系运算理论，才能深刻理解查询语言的本质并且熟练使用。实际上，曾经分别产生过基于“关系代数”和“关系演算”的关系数据操作语言，现有 SQL 同时基于“关系代数”和“关系演算”。

作为数据库发展历史上最为成功的数据操作语言，SQL 具有突出的特点和明确的优势，如简单易用、非过程化和与宿主语言的有效整合等。SQL 最基本的功能是数据管理，其中主要有数据定义（数据库模式、基本表和视图、存储过程和触发器等的创建、修改与撤销）、数据查询（选择、投影与连接）及数据更新（插入、删除与修改）。另外，SQL 还具有数据操纵功能，如关系数据的完整性维护与安全性保护、并发控制与故障恢复等。需要指出的是，SQL 与宿主语言的有效整合——嵌入式 SQL 和动态 SQL 在数据库应用系统开发过程中具有重要的作用。

数据操作（定义、查询和更新）是所有关系数据库都必须具有的基础功能，而数据操纵（数据库保护和事务管理）则属于关系数据库的高层次机制。数据库保护就是数据库的完整性和安全性保护，完整性包括数据的正确性和一致性，其具体技术实现就是要维护关系数据的实体完整性、参照完整性和用户定义完整性；安全性是为防止对数据库的恶意攻击和破坏，从数据库角度而言，“授权”机制是关系数据安全性的基本保护措施。实际应用通常都需要进行多次数据操作才能完成，在技术实现层面，这“多个”操作需要作为一个“整体”对待，这里“整体”的含义就是“原子性质”：多个操作“要么全做，要么全不做”，这就引入了“事务”概念。“事务”概念的意义在于它为关系数据库的“并发控制”和“故障恢复”提供了一种有效的技术实现框架，在此框架下，“并发控制”可以通过“封锁”机制实现，“故障恢复”可以通过“备份与日志”机制实现。

主要参考文献

[1] E F Codd. A Relatioanal Model for Large Shared Data Banks[J]. Communications of the ACM. 1970,13(6): 377-387.
[2] 王珊，萨师煊. 数据库系统概论[M]. 4 版. 北京：高等教育出版社，2006.
[3] 施伯乐，丁宝康，汪卫. 数据库系统教程[M]. 北京：高等教育出版社，2003.
[4] 王能斌. 数据库系统教程(上册)[M]. 2 版. 北京：电子工业出版社，2010.
[5] 徐洁磐. 现代数据库系统教程[M]. 北京：希望电子出版社，2002.
[6] 尹为民，李石君，曾慧，等. 现代数据库系统及应用教程[M]. 武汉：武汉大学出版社，2005.
[7] 郑若忠，宁洪，等. 数据库原理[M]. 北京：国防工业出版社，1998.
[8] 黄德才. 数据库原理及其应用教程[M]. 北京：科学出版社，2002.
[9] Abraham Silberschatz, Henry F Korth, S Sudarshan. 数据库系统概念[M]. 5 版. 杨冬青，唐世渭，等译. 北京：机械工业出版社，2007.
[10] Date C J. 数据库系统导论[M]. 孟小峰，王珊，等译. 北京：机械工业出版社，2000.
[11] J D Ullman, J Widom. 数据库系统基础教程[M]. 岳丽华，龚育昌，等译. 北京：机械工业出版社，2006.

［12］ Ramez Elmasri，Shamkant B Navathe. 数据库系统基础［M］. 邵佩英，张坤龙，等译. 北京：人民邮电出版社，2002.
［13］ Date C J. 深度探索关系数据库［M］. 熊建国，译. 北京：电子工业出版社，2007.
［14］ Date C J. SQL 和关系数据库理论［M］. 周同兴，等译. 北京：清华大学出版社，2010.
［15］ 叶小平，汤庸，等. 数据库系统基础教程［M］. 北京：清华大学出版社，2012.

第3章 面向对象数据库

随着计算机技术发展和应用需求驱动,面向对象数据库(Object-Oriented DataBases,OODB)受到人们的广泛关注。OODB技术建立在面向对象思想方法基础之上,以将静态属性与动态操作作为封装体的"对象"作为数据元素,以同型对象的集合"类"作为数据存储与操作的基本单元,更加符合人们对于客观世界的直观认识,对现实世界中复杂事物具有较强的语义描述能力,同时也具有高效开发应用系统和实现软件重用的功能,能够在传统数据库技术难以有效支持的领域获得成功应用。OODB研究始于20世纪80年代,当时主要是针对关系数据库的一些弱项,如有限的数据类型、没有基于系统的全局标识、不支持用户或系统可扩充的函数,以及运算和难以处理的复杂数据对象等。虽然人们在OODB技术的概念和原理上还未取得完全一致的理解,但大都认为对象、类、继承和封装等是OODB应该具有和支持的基本元语。进入20世纪90年代,从事研制面向对象数据库管理系统的厂商组成了ODMG集团(Object Database Management Group,ODMG)以着手制定OODB标准,并于1993年8月公布第一个ODBMS工业化标准——ODMG—1993。此后,又相继推出ODMG 1.2、ODMG 2.0(1997)和ODMG 3.0(2000)。本章将简要介绍基于ODMG 3.0的OODB技术。

3.1 数据管理新的需求

自20世纪80年代以来,关系数据库(Relational DataBases,RDB)在传统商业事务处理领域获得巨大成功,随之带动了其他各领域对数据库技术应用需求的巨大增长。这些领域主要包括计算机辅助设计、计算机辅助软件工程、计算机集成制造系统、办公室自动化系统、地理信息系统、知识库系统、数据仓库与知识发现、多媒体系统和计算机网络系统等。这些新领域所需要的数据管理功能相对于传统情形具有许多新的特点。

(1) 存储和处理复杂对象。数据对象内部结构复杂,相互联系具有多种语义,难以用常规关系结构进行描述与表达。

(2) 多样数据类型。从数据逻辑描述来说,数据项取值可以是半结构或无结构数据类型;从数据应用背景来说,有时态和版本数据类型、空间数据类型、时空数据类型及各种用户自定义数据类型等。

(3) 数据常驻内存。数据常驻内存中包括数据管理及对大量数据对象的存取和计算。

(4) 数据处理与程序语言集成。由于数据结构的复杂化和实际应用背景的不断拓展,需要将常规程序设计语言与数据库语言进行无缝整合才能完成各类数据管理任务。

(5) 支持复杂事务管理。支持长事务和嵌套事务处理。

面对新的应用需求，RDB 的弱项或缺陷日益显现出来。

(1) 数据类型简单。RDB 只能表达、存储和处理确定的某些简单数据类型，不易根据应用需求扩展数据类型集合。对于各种复杂类型依赖用户自行编写的应用程序进行个别构造、描述和处理，这显然与数据库管理数据初衷不相符合。

(2) 结构与行为分离。RDB 主要关注数据独立性和存取效率，语义表达能力较差。数据实体结构存储在数据库当中，而数据实体行为常常需要由应用程序另外实现。这种结构与行为分离导致数据库中复杂结构数据只能通过相关应用程序进行解释执行，限制了用户对数据的有效使用。

(3) 查询实现复杂 RDB。由于规范化的基本要求，通常需要将复杂对象分解为多个基本关系表进行处理，逻辑结构与外部实用场景差别较大，由此建立在相应逻辑结构之上的查询实现相当复杂。DBMS 需要在实现数据存储透明和查询优化方面付出沉重开发代价。

(4) 阻抗失配。代价之一就是“嵌入式”SQL。由于 SQL 语言面向集合，基于嵌入式应用程序语言面向记录，两者编程模式不同，类型系统各异，需在相互之间通过适当方式进行转换，即会出现“阻抗失配”。解决“阻抗失配”需要使用较多的应用程序，增加了用户负担和系统开销。

针对 RDB 的不足及为支持在新领域中的应用，经过实践对比，人们发现面向对象技术与数据库技术相结合而形成的对象数据库具有广阔发展前景。

首先，在物理和逻辑两个层面上，通过面向对象技术可对常规面向记录的数据格式进行复杂语义结构描述，从而大大扩展了数据库技术的应用范围。

其次，数据管理技术的复杂化意味着相应安全性要求的进一步提升。基于对象数据库能够以更为自然的方式在抽象层面上进行结构和行为封装的复杂对象建模，在降低用户使用复杂度的同时，也为复杂数据处理过程“自动”提供了可靠的安全性保障。

基于对象数据库技术研究始于 20 世纪 80 年代中期，现在已经取得了很大进展，成为数据库发展的一个重要方向。通常认为，对象数据库系统应当具有下述基本功能。

① 数据模型：基于对象、封装、继承和多态等概念原理进行建模。

② 数据语言：支持消息传递，提供具有完备计算能力的数据库语言。

③ 数据管理：提供持久化对象和长事务处理能力。

④ 数据相容：兼顾传统关系数据管理。

最后，从对象数据库技术发展实践来看，通常有着下述 3 种研发路线。

① 基于 RDB 的面向对象扩充：基本点是建立能够适应现实需求数据类型系统。

② 基于对象程序语言的数据库扩充：基本点是基于 C++或 Java 等建立相应数据库语言管理数据库数据。

③ 全新对象数据库技术：创建不依赖于现有任何技术的对象数据库系统。

沿着第一个方向，可以建立对象关系数据库(Object Relational Data Bases，ORDB)系统。例如，PostGreSQL 就是一个对象关系数据库系统(ORDBMS)，同时一些代表性的关系数据库也支持对象关系模型，如 Oracle 数据库等；沿着第二个方向，可以建立 OODB (Object-Oriented Data Bases)系统，如 ONTOS、ORUON 等，它们均是 C++的扩充，掌握 C++的人们都可以方便地使用这类数据库系统；而沿着第三个方向发展，系统全新，难度很

大，迄今没有太大进展。

3.2 阻抗失配与对象持久

如前所述，OODB 技术主线是将面向对象程序设计语言（如 C++或 Java）进行数据库意义下的扩展，使其成为一种有效的 OODB 语言。程序设计语言和数据库语言存在着较大差异，欲将两者整合，首先需解决由于相互间差异而带来的“阻抗失配”和“对象持久”问题。

3.2.1 数据库语言与程序语言差异

数据库语言和高级程序设计语言之间由于基本着眼点与处理过程格调的不同而具有较大差异，将两者整合起来以形成一种面向对象数据库的语言，需要根据两者差异而采用相应的技术措施。

1. 查询语言与程序语言的阻抗适配

经典数据库查询语言（如 SQL）是由系统自动选择路径的非过程性语言，同时具有面向集合的操作方式，这就与高级程序设计语言（如 C++）面向单个数据的过程性操作方式之间出现不易协调的问题，通常称为“阻抗不匹配”或“阻抗失配”。在 RDB 技术中，嵌入式 SQL 的着眼点就是要解决这种不可避免的阻抗失配问题，但此时也带来两个不容忽视的问题。

（1）“对象”和“元组”之间的技术转换在类型系统之外进行，系统难以进行有效控制，从而会增加出现未能及时检测到的转换错误风险及数据安全性问题。

（2）“对象”和“元组”之间的技术实现需要大量程序代码支撑，相应转换过程会占用相当多的系统资源，从而会对提升系统运行效率产生较大影响。

在 OODB 技术当中，解决阻抗不匹配的基本思路是将数据库查询语言完全集成到面向对象程序设计语言当中，共用相同的类型系统，使得对象在数据库创建和存储过程中不需要任何显式的类型与格式转换，从而做到任何所需转换的实现对于用户来说都是透明的。这种将“查询”无缝集成到“操作”过程中的技术路线就是将相应的宿主语言（如 C++）进行数据持久化，或者说将对象持久化。

2. 持久化程序设计语言

面向对象程序设计语言（如 C++）的基本元语是对象和类的概念及它的若干结构关联，但此时对象具有“瞬时”性特征，即对象只在程序执行期间存在于非永久性存储器（缓冲区或主存）当中，在程序执行完成后即可消失，就像 C++中各类变量在程序结束后立刻消失一样。

数据库作为保存数据的电子容器，其基本要求就是数据具有“持久”性，即当创建数据和操作数据的程序停止之后，所得到的结果数据依然需要驻留在计算机（外部存储器）当中。

在 OODB 中，对象作为数据应当是持久性对象，需要保存在永久存储器（磁盘）中，它不仅在程序执行过程中存在，程序结束后还必须存在。因此，将数据库查询语言集成到程序设计语言的技术实现就需要以一种持久化程序设计语言为基本出发点，这通常都是对某种对象程序设计语言（如 C++或 Java）进行对象持久化扩充而展开研发。

3.2.2 对象和对象标识持久化

OODB 中的对象持久化包括对象自身持久化和对象标识符持久化。

1. 对象持久化

对象持久化通常可以通过下述途径实现。

(1) 按类持久。直接声明所创建的一个类具有持久性,这在逻辑意义上就意味该类所有对象实例在默认情况下都是持久对象,而未加以持久声明的各个非持久类中对象实例都是瞬时的。但在实际应用中,一个类中的对象实例中可能存在需要具有持久性和不需要具有持久性的多种情形,此种"按类持久"方式显得不够灵活。

(2) 按对象持久。为处理"按类创建"灵活性欠缺问题,采用"按对象持久"技术,此时可分为以下两种情形处理。

① 创建时持久:扩展创建瞬时对象的语法为新的用于创建持久对象的语法。在对象创建时,如果是按照持久对象语法定义的,该对象在相应过程中就为持久对象;否则就是"瞬时"对象。这是一种在创建时就确定对象是否持久的方式。

② 应用中持久:在创建时,所有对象都按照程序设计语言中原有机制定义为瞬时对象,而当一个对象在程序结束后需要持久存在时,就在程序结束前显式地给予适当标志而将其标识为持久对象。这是一种将决定对象持久与否推迟到对象创建之后的方式,在实用中会更具灵活与方便。

(3) 按可达性持久。将一个或多个对象声明为(根)持久对象,而对于其他对象,则根据相关继承或组合语义考查其能否从一个根持久对象直接或间接引用来决定持久与否。此时,所有由"根"持久对象引用的对象都是持久的,而这些持久对象的再次引用也是持久的,即持久对象到根持久对象是可达的。这种方式在理论上只需声明根结点对象持久即可,由相应语义树就可以方便地得到所需要的持久性数据结构;但在技术实现方面,由于按照引用路径判断某个对象是否持久实际上是一种"遍历"查找过程,可能需要较高的系统开销代价。

2. 对象标识持久化

在面向对象程序设计语言中,创建一个对象时,系统返回一个瞬时对象标识符,该标识符只在程序执行过程中有意义,程序结束,对象删除,相应标识符也就消失。因此,在创建持久对象的同时还需要赋予对象一个持久性标识符。注意到在 C++ 中,瞬时标识符实际上就是一个内存指针,而一个对象与其存储空间的关联可能会随着时间而发生变化,为应对这种变换,对象标识持久程度可分为以下 4 种情形。

(1) 过程内持久标识符。标识符只在单个过程执行期间内具有持久性。例如,过程内一个局部变量的标识符就是如此。

(2) 程序内持久标识符。标识符只在程序或查询执行过程中持久。例如,一个程序中全局变量的标识就只在该程序运行中有效。

(3) 程序间持久标识符。标识符只在一个程序执行到另一个程序之间持久。指向磁盘上文件系统数据指针提供了程序之间标识,当数据在文件系统中的存储方式发生改变时,其相应标识可能也会发生改变。

(4) 持久标识符。标识符不仅在各个程序执行期间,而且在数据结构重组期间都是持

久的。OODBS 主要考虑持久标识符。

在 C++语言的持久化拓展中，持久对象的持久标识符通常使用“持久化指针”，其在程序运行结束后依然有效，同时也跨越了某种形式的数据重组，用户在使用 C++过程中可以像使用内存指针一样使用持久化指针，在逻辑上可将持久化指针看作是指向数据库中对象数据的指针。

3.2.3 持久对象存储和查询

OODB 需要存储持久对象。持久对象进入数据库时，就要将其标识转换为一个永久的唯一 OID，由此才可以使得该对象在数据库中真正“持久”。如果需要将该对象由数据库取到内存，则要执行相反操作，即当某个对象读入内存后将其相应 OID 类型指针所指向的逻辑磁盘地址转换为主存中的实际存储地址。

1. 持久对象的存储

从逻辑上来看，类中对象实例的数据部分（如对象的属性和关联）需要针对相应的单个对象进行单独存储，类中方法代码则需要作为 OODB 数据模式重要组成部分而统一存储，但两者都是存在同一个存储器系统当中，这就类似于 RDB 中将关系数据表和相应存储过程都存储在一个系统当中。在实际处理时为了避免将程序编译器这样大规模系统集成到数据库当中而使得系统结构变得更为庞大和复杂，通常在技术实现上将类方法代码存储在与数据库分离的适当文件系统当中。

2. 持久对象的查询

OODBS 在查询对象数据查询过程中提供下述 3 种数据定位方式。

（1）根据对象名称访问对象。每个对象分配有一个如同文件名那样的名称，系统通过对象名称定位所需要查询的对象。对象的命名和对象名称管理需要占用较多系统资源，因此该方法在数据库中对象数量较小时比较有效，而当对象数量较大时就难以适应。

（2）根据对象标识查找对象。提取对象标识符或持久化指针并将其存储在与数据库分离的外部文件当中，系统根据对象标识符定位对象数据。与对象名称不同，对象标识不必具有用户记忆性，甚至可以作为指向数据库内部的物理指针而存在。

（3）根据对象集合体检索对象。将对象存储为一个诸如集合（set）、多集（multiset，可具有相同元素值的集合）和列表（list，具有不同元素值的有序集合）的集合体，并允许使用程序在其上反复循环搜索所需对象。集合体的一个常用形式是“类外延”（class extent），这是一个属于给定类的所有对象之集。当对一个类定义了类外延之后，创建具体对象后，该对象会被自动插入到类外延当中；而当对象被删除时，就会自动将其由类外延移出。类外延的意义在于可以对各个对象定义语义标识——键（key），从而可以像看待关系表那样看待类，像检查一个关系表中所有元组那样检查类中的所有对象类。

大多数 OODBS 都支持上述 3 种数据库访问方法。通常对涉及的所有对象都赋予对象标识符，只对类外延或集合体中及其他需要特定选择的对象赋予对象名称，而对绝大多数对象都没有给予对象名。在实际应用中，多数系统只对持久对象维护相应的类外延。

3.2.4 面向对象数据模型

面向对象数据模型（Object-Oriented Data Model，OODM）是基于面向对象思想与方法构建的数据库逻辑模型，可以通过面向对象数据结构、建立在数据结构之上的数据操作和基

于数据结构的相关约束予以表示和说明。

(1) 数据结构。数据结构主要是指对象和类自身原有的内在结构,如对象的属性与方法封装、类的基于继承机制的层次结构和基于组合机制的复合结构,以及类方法的重载与联编等。实际上,人们研制对象数据库数据的初衷就是注重这种远比原有数据库中数据结构(格式化、关系型的逻辑结构及 E-R 甚至 EE-R 的概念结构等)具有更为强大语义表达能力的面向对象数据结构形式。

(2) 数据操作。类中定义的各类方法就构成了基于对象数据结构的数据操作。由于这些操作是采用功能强大的面向对象程序设计语言编写,相比于常规的数据库操作实现,可以处理更为广泛和更为复杂的数据应用问题。

(3) 数据约束。从本质上来看,数据约束也是一种方法的表示,因此可以通过定义和调用适当的类方法实现相关的数据约束。

由此可见,从理论上来讲,OODM 实际上就是对面向对象程序设计语言中各类元语和重要机制进行适当的数据库语义解释。现今一种通行的规范化语义释义和机制重编就是 ODMG 标准。

3.3 对象和类的数据库释义

如前所述,OODM 从形式上来看就是“借鉴”面向对象方法中的基本概念和机制,按照数据库数据模型要求进行“重编”。由于数据库语言和程序设计语言自身的差异,特别是数据库中数据的“持久性”和程序设计语言中数据的“瞬时性”存在的鸿沟,两者整合的前提就是程序设计语言的持久化问题。此外,由于数据库语言与程序设计语言原始出发点和技术格调的不同,需要按照数据库语义对面向对象方法中的基本元语和重要机制进行必要的“释义”。这种释义首先要规范严明以便为人们所普遍接受;其次要面向 OODB 的技术实现,为此在技术层面上要有明确的数据库技术指向。基于 ODMG 的 C++标准就是这样一套对象元语的数据库释义规范。

3.3.1 ODMG 标准与核心概念

面向对象数据建模多采用基于面向对象程序设计语言的持久性扩充。面向对象程序设计语言主要有 Simula、Smalltalk、C++和 Java。其中,形成面向对象数据建模标准的主要是基于 C++的 ODMG 标准。

1. ODMG 标准

成立于 20 世纪 80 年代的 ODMG(Object Data Management Group),在 1993 年颁布了 OODB 工业化标准——ODMG1.0,完成了对 C++的数据库功能扩充;接着,又在 1997 年和 2000 年分别推出 ODMG2.0 和 ODMG3.0。ODMG 标准主要由以下 3 个部分组成。

(1) OSL。OSL 又可分为对象定义语言(Object Definition Language,ODL)和对象交互格式(Object Interchange Format,OIF)两个组成部分,其中 ODL 作用类似于关系数据库中的 DDL 语言,用于创建数据库对象;OIF 基本功能是为 ODL 所创建的对象类型快速创建相应对象实例,同时还可以为对象实例赋予初值。

(2) OQL。OQL 为对象数据查询语言,与 SQL 类似,用于对象数据基本操作。作为数

据库系统与用户之间的主要接口，OQL 没有 SQL 中的数据更新语句，这是由于数据更新可以看作是方法，在 OODB 中，这些功能都由 ODMG 语言绑定实现。

(3) 宿主语言绑定。在过程化程序设计语言内部使用 ODL 和 OQL，实现 C++、Smalltalk 和 Java 绑定。

需要说明的是，ODMG 已从 2002 年开始就停止了继续发展。这说明 OODB 自诞生之日起一直都没有能够取得像 RDB 那样的商业成功，但这并不意味着 OODB 的失败，实际上，对于各类新型数据如网络数据和多媒体数据等持久性管理存储和操作运算的需求依然存在并且还有增长，很多具有高复杂度和高性能要求的数据应用需要运行在 OODB 之上，此时采用基于 SQL 的 RDB 会付出很高的性能代价。特别是 ODMG 将数据库语言和程序设计语言这两种有着很大差异的语言进行有效整合，这为数据库技术新发展在理论和技术上提供了成功先例和基本借鉴。这也可以看作是人们将 OODB 视为继 RDB 之后第三代或新一代数据库重要代表的原因所在。

2. ODMG 元语

划分数据库中数据的基本粒度是建立相应数据模型的基本支撑。例如，在关系数据模型(RDM)中，其数据的逻辑粒度为元组、关系表和数据库文件，而其数据的技术处理粒度为属性项、元组、关系表和数据库文件。对于对象数据模型(OODM)来说，其数据的逻辑粒度分别为对象、类和数据库文件，而数据技术粒度为状态(属性和联系)、操作(方法)、对象(文字)、类和接口等。同时，数据模型的核心是所涉及数据之间的相互关联，即数据结构，RDM 在不同关系表中元组之间的关联是双向 $m:n$ 联系，不同关系表之间是"外键"联系；对于 OODM，不同类中对象之间的关联也是双向 $m:n$ 联系，而不同类之间则主要是"继承"和"组合"联系，以及由此而产生的面向对象精髓"多态"机制。基于 ODMG 工业标准，OODM 需要在数据库的语义框架中对数据元素"对象"，以及"类"和相应数据关联进行必要的逻辑界定和技术处理。ODMG 中主要有下述 5 个核心概念。

(1) 对象和文字。从技术处理角度来看，ODMG 进一步将常规逻辑语义上的"对象"明确区分为"对象"和"文字"两种情形。

① 对象：具有唯一标识符，无论对象是存储在外存中还是在内存中，相应对象标识符在对象整个生命周期中都有效。作为基本数据结构，对象是存储和操作的基本单元。

② 文字(literal)：没有标识符，一般不能单独作为数据粒度存在，需要嵌入在相应对象当中作为对象成分(如属性值等)使用，使用过程中文字值通常也不能改变。

(2) 类型。具有相同特征和方法的对象，以及具有相同特征的文字都可以被划分为类型(type)，同一类型的对象或文字具有相同的描述形态。在 ODMG 中，类型定义通常由一个外部声明和若干实现组成。

(3) 状态。由于数据查询的需要不同于常规对象情形，数据库语义下的对象静态性质除了对象的属性外，还需要添加不同类之间及对象之间双向的"$m:n$"联系。"属性"(attribute)和"联系"(relationship)就构成了数据库意义下对象的静态特征描述，这在 ODMG 中称为对象的状态(states)。

① 属性：属性不看作对象，作为单一类型，它没有对象标识，也不定义属性的属性或属性之间的联系。

② 联系：联系定义在两个不同类型(对象集合)之间，是一种双向的二元联系(有 3 种联

系方式分别为 1∶1、1∶n、m∶n)。由于是双向联系,因此数据创建中的联系需要成对出现。另外,联系需要具有对象标识可引用的实例。由于文字没有对象标识,因此不能出现在联系当中。

(4) 操作。对象行为通过一组操作(operation)定义。由于需要考虑到类之间的继承与组合,因此定义对象行为时还需要考虑操作的多态机制,其中包括覆盖和重载等基本技术。需要注意的是,操作需要与一个且只能与一个类型相关联。操作具有输入和输出参数,并且可以返回特定类型的结果。

(5) ODL。对象定义语言。系统利用 ODL 定义对象数据管理系统的模式,它所存储的对象都是模式中定义类型的实例,即对象,这些对象可以供多个应用程序共享。

以下分别讨论对象和类的基本构成。

3.3.2 对象与文字

对象是面向对象思想中最为基本的原始概念,也是一个伞形概念。ODMG 标准对其进行完整的数据库语义解释。

1. 对象概念

面向对象方法中,对象是一个将静态属性和动态方法整合的封装体,通过消息与外部进行关联。

1) 对象

ODMG 中的对象由变量(属性)、函数(方法)和消息 3 个特性组成。

(1) 变量:包含对象数据,相当于 E-R 模型中的属性,也称为对象的状态。

(2) 函数:实现一个消息的代码,一个方法返回一个值作为对消息的反馈,也称为行为或方法。

(3) 消息:对象所能响应的每个消息具有若干参数,对象接受消息后应做出消息的响应。

对象的属性和行为(方法)需要进行封装(encapsulation),对象之间的相互作用通过发送和执行消息完成,以此保证使用者只能看到对象封装界面上的信息,对象内部对使用者隐蔽,其目的在于将对象使用者和对象设计者分隔开来。

在 ODMG 标准中,同时考虑对象的临时(transient)和持久(persistent)两种生存期(lifetime)。如前对象持久性所述,对象生存期可以根据实际情况采用灵活方式进行瞬时性或持久性确定。在 RDB 中,使用 SQL 定义的关系实例都是持久的,使用高级编程语言定义的类型都是临时的,而且通常使用 SQL 来定义和处理永久数据元素,使用高级程序设计语言定义和处理临时数据元素。在 ODMG 中,临时对象与持久对象创建与处理至少在逻辑层面上对于用户都是透明的。

2) 对象标识

每个对象都应具有一个对象标识符(OID),这是对象在系统范围内的唯一标识,其意义有以下两点。

(1) 在实际存取过程中区分硬盘中同一个存储区(storage domain)中的各个对象。

(2) 在类的继承和组合关系中实现间接调用,提升系统效率和安全性保障,避免逻辑循环。

对象标识符在对象创建时由系统自动产生，在概念层面上与用户定义的对象状态和操作相互独立，也与具体应用无关。对象标识一旦生成，即使对象状态发生变动也不可改变，以保证状态改变前后都看作是同一对象。用户可以查询对象标识，但不可改变对象标识的具体取值。

对象还可拥有名称(Name)，应用程序可以使用对象名称来引用对象，系统将对象名称映射到对象标识符，进而定位到相应对象。对象名称不同于对象的键，不在对象所属类型的语义范畴，从而在系统中通常也不具有唯一性。一般而言，名称用作数据库中数据语义入口，系统通过对象名称定位对象。管理大量名称将消耗系统资源，因此不是所有对象都需要赋予名称。

3）对象状态

从组成上来看，对象具有 3 个基本要素，即属性、操作和联系。其中，属性和操作表明单个对象的特质，联系描述不同类中各个对象的关联。数据结构实际就是数据之间的关系。在 OODM 中，由于数据的逻辑粒度为对象和类，因此数据之间的关系也分为对象之间的联系和类之间的联系两个层面。类之间的联系主要是类的继承与组合关系，这可以看作是“模式”之间的关联；对象之间的联系实际上就是关系模型中涉及“互逆”遍历的 $m:n$ 联系，而这种基于双向遍历的关系对于数据查询非常必要，因为数据查询的一个基本要求就是查找满足查询要求的所有数据而不是其中的个别或某些数据。如前所述，在 RDM 中，这种遍历关联是和属性描述分开处理的，但在 OODM 中，由于两者都属于数据的静态性质，为了与操作(方法)这种动态性质相区别，ODMG 标准中将其统称为状态(state)。从类型观点来看，属性和联系都是“状态”类型的两个子类型。

① Attribute 子类型：描述对象属性，属性取值都是文字。

② Relationship 子类型：描述不同类型或同一类型中非文字对象之间的二元关系，表示基于 is associated with 的引用(reference)语义，刻画了对象中的一种“相等”关联(equal association)。联系参与者是对象，因此联系取值只能是 OID 及其集合。

例如，在学生与课程两个类型的对象之间存在一种选课 take 和被选 taken-by 的 $m:n$ 联系。这种二元联系可定义为：

```
interface Student
( …
relationship Set(Course)take inverse Course::taken－by,
… )
```

以及

```
interface course
( …
relationship Set(Student)taken－by take inverse Student::take
… )
```

上述定义中，take 的类型为 Set(Course)，即某位学生所选课程的 OID 集合。taken-by 的类型为 Set(Student)，即选修某门课程所有学生的 OID 集合。通过 take 可使用导航式方式查询一个学生的所有选课；通过 teken-by 可使用导航方式查询选修某一门课程的所有学生。这里，通常将 take 和 taken-by 称为联系的遍历路径(traversal paths)。

如果分别单独设立 take 和 taken-by 两个互逆遍历路径，有可能带来一致性维护方面的问题。当某学生退选一门课程时，不仅需要修改对应学生对象中 take 遍历路径，还需要修改相应课程对象中 taken-by 遍历路径。如果只修改一处，就会引起不一致。为此，在说明 take 时，使用关键字 inverse 说明在 Course 类型中还存在一个 taken-by 的逆遍历路径。对于 taken-by 也是如此。有了这样的说明之后，无论是从任何一方进行修改，系统都可以自行维护数据的一致性。Relationship 通过导航方式实现了不同类型对象之间的相互联系，完成了对于复杂对象的计算，是避免开销很大的连接运算从而提高系统效率的基本技术。

2. 原子对象与构造对象

对象作为一个伞形概念，可以具有属性、联系和方法 3 个组成部分（要素），但不必同时具有这 3 个要素，实际上，只需具有其中之一就可以称之为逻辑意义上的"对象"。但从技术处理角度考虑，同时具有三要素和只具有三要素之一或之二的对象却具有较大差异，为描述和处理这种差异，ODMG 标准中规定，如果一个对象同时具有"对象三要素"则将其定义为原子对象（结构对象），否则就定义为构造对象（非结构对象）。

① 原子对象：完整具有"属性""联系"和"操作"3 个组成部分的对象，对象本身不可再做进一步分解。

② 构造对象：部分具有"属性""联系"和"操作"3 个组成部分的对象，对象内部还可以进一步分离。

明确原子对象和构造对象的意义在于分别建立不同的对象构造器函数。

1）原子对象

原子对象（atom object）也称为"用户定义对象"，是具有完整"三要素"的对象。具体而言，所有非聚集对象，以及由用户定义的具有属性、方法和联系的对象都是原子对象。原子对象实际上就是具有"整体化"结构的对象。例如，Faculty 对象有一个带有许多属性、联系及操作的一体结构，3 个部分的任何一个缺失或分离后都不足以确定 Faculty 的完整语义，因此是一个原子对象。用户定义的"同型"原子对象类型可被定义为一个类，以统一刻画描述类中原子对象具有的属性、联系与方法，如用户可以为 Faculty 对象指定一个类。

原子对象及其相应类是"纯正"的逻辑意义下的 OODB 数据元素，而非原子对象及其相应类是在"统一"的观念下"看作"或"处理为"OODB 意义下的数据元素。需要注意的是，ODMG 中不存在系统预置的原子对象。

2）构造对象

构造对象（structured object）是指对象三要素有所缺失的对象，在实际应用中，通常是具有属性而联系或方法缺失，或者两者都缺失的数据对象。这里与 C++ 情形类似，是为了兼容常规意义下的复杂数据类型，如"聚集"与"结构"等情形。

（1）聚集对象（collection object）。在某种意义上可看作 C++ 中数组在数据库框架下的扩展，作为一个带有参数的类型生成器，聚集对象中所有组成元素的类型可以记为 T。聚集对象本身可看作是一个具有某些附加条件的集合，其中所有元素都属于同一类型，这些类型可以是原子对象、其他聚集对象和文字类型。

ODMG 支持以下 5 种聚集对象类型（集合体）。

① 数组（Array < T >）：元素按一定格式排列，可以重复，每个元素可以按照其附有的下标进行访问，数组容量大小可以动态变化。

② 列表(List＜T＞)：元素有序但元素取值没有重复。在列表类型中，可以从表头、表尾或指定位置读取、替换、插入和删除元素。

③ 包(Bag＜T＞)：包中元素不排序且允许重复，也称为多集(multiset)。

④ 集合(Set＜T＞)：元素不排序且不允许元素重复。

⑤ 字典(Dictiongary＜T,S＞)：设 T、S 是任意类型，则字典类型 Dictiongary＜T,S＞表示 T、S 有序对的有限集合，其中，T 称为键类型，S 称为值域类型，由此可知字典类型 Dictiongary＜T,S＞中的元素就是(键，值)二元组，其基本要求是各个二元组不能完全相同。字典类型允许关联查询，即给定键值就可得到一个特定关联对，这与索引类似。下面是每个(键，值)二元组都是结构 Association 的实例。

```
struct Association{
    Object key;
    Object value;
};
```

ODMG 中分别定义了数组工厂、列表工厂、包工厂和集合工厂来创建聚集对象，它们都是对象工厂的子类型。这些聚集工厂接口的定义分别简写为：

```
Interface ArrayFactory:ObjectFactory{
    …
};
Interface ListFactory:ObjectFactory{
    …
};
Interface BagFactory:ObjectFactory{
    …
};
Interface SetFactory:ObjectFactory{
    …
};
```

其中，“…”为系统内置内容，详细可参考有关资料。

(2) 结构对象(Structural Object)。结构对象的组成元素通常具有不同的数据类型，这不同于聚集对象的情形。结构对象可以类比于 RDB 中的元组和对象关系数据库中的行(结构)类型。

3. 文字

对象需要有标识。创建和管理对象标识是一项较大的系统开销。对象标识的基本作用在于区分对象，但在实际应用中，有许多对象特别是非结构对象并不需要进行特别区分，数据本身的字面值就可担当其标识作用，因此可“省略”对象标识。没有专门对象标识的对象就是 OODM 中文字。

1) 文字基本性质

文字(literal)是指没有对象标识的一个对象实例或对象值。相对于变量，文字相当于常量。ODMG 中的文字具有如下特性。

(1) 文字语义。文字语义直接由表示它的字符确定而无须另外释义，即“见其面而知其值”。例如，由“1949-10-01”就直接可以知道其语义为“中华人民共和国成立日”。

(2) 文字标识。文字自身的值就是其标识而无须专门创建相关 OID,但可为文字赋予名称。

(3) 文字修改。如上所述,文字取值更新后其语义特别是标识都会相应改变,从对象角度考虑,就不再是原先文字(对象),这相当于撤销原先文字而重新设置新的文字,从这个意义上而言,文字具有不可更改性(immutable)。

2) 原子文字

原子文字(atom literal)是指预定义的并且对应于基本数据类型的对象值。例如,C++中整数、实数及各类合法字符都是常用的原子文字。

在 ODMG 标准中,原子文字主要有 long、long long、short、unsigned long、unsigned short、float、double、boolean、octet、char、string、enum 13 种。

其中,枚举类型(enum)是一个类型生成器,定义枚举类型时需要说明取值范围。例如,当属性 gender 被定义为枚举类型时,取值范围为{male,female}。

```
attribute enum gender{male,female};
```

3) 构造文字

构造文字(structured literal)分为持久聚集(immutable collection)文字和持久结构(immutable structure)文字两种子类型情形。

对象和文字都可以有自身的构造子类型,但两者的意义有所不同。

(1) 基于对象构造子类型:完整意义下的对象,具有对象的各种功能。

(2) 基于文字构造子类型:只是一个常量,没有属性、联系和操作说明,没有 OID 也不能更改。

由此可知,构造文字实际上就是一种数据常量而且不可更改,因此,本体上就具有持久性,这种基于聚集和结构的对象形式通常也称为"持久聚集"和"持久结构"。

ODMG 支持的结构文字包括 Date(日期类型)、Interval(时间间隔类型)、Time(时间类型)、Timestamp(时间戳类型)4 种。

ODMG 提供上述类型的类型生成器 struct,也可以在应用程序中自定义结构文字类型,如下面的 Passport 就是应用程序自定义的结构文字。

```
struct Passport {
    string passportNum;
    string nationality;
    date issueDate;
    date expiryDate; };
```

3.3.3 类型、类和接口

类和对象一样,是面向对象思想中最重要的元语,需要结合数据库技术实现的指向对它们进行较为精细的释义。

1. 类型和类

"类型"和"类"是 ODMG 的常用语。在面向对象程序设计语言当中,类型(type)和类(class)常常通用。例如,在 C++中说明一个对象属于某个类和该对象属于某种类型往往具

有相同的含义。但 ODMG 标准对“类型”和“类”进行了适当区分。

(1) 类型：具有所涉及方法及该方法的所有实现。

(2) 类：具有所涉及方法及该方法的一种或几种实现。

由此可知，类型不依赖于其中方法的具体实现，即对于方法实现透明；类依赖于其中方法的具体实现。也就是说，ODMG 中“类型”抽象层次高于“类”，类型可以概括类。而在所涉及方法有且仅有一种实现方式时，“类型”和“类”才在逻辑上具有相同语义。当然在实际应用中，方法多是同其实现整合在一起的，因此，如同在程序语言中那样，在 ODMG 表述中，“类”的概念一般更为常用。

正是由于类型具有较高的抽象层次，因此类型中可以有实例也可以不定义实例。作为具有相同静态和动态性质的对象实例的集合，类通常是不能没有对象实例的。由此可知，“类”相当于 RDB 中的满足适当条件的同型元组的集合，此时，这种“适当条件”就是关系表的域约束（关系外延）。在 RDB 中，数据对象的域约束对于有效的数据查询至关重要。为此，通过前述提及的类外延也就将常规数据库中的域约束概念拓展到 OODB 当中。

类的一个基本情形是可以定义相应的类外延的，而类外延实际上就是将类中对象作用域显式化，其意义在于进行类的创建时，能够先行确定其对象实例的范围以利于数据的有效查询和管理。

“类外延”在 ODMG 标准中也简称为“外延”(extent)，即类的所有持久化对象实例全体构成其外延，并将其看作是类的一个组成部分。每个类的外延由系统自动创建并加以维护。可对其进行命名以供用户访问。外延是类外延（持久化对象集合范围）的数据库释义，其意义在于以下两点。

(1) 通过键进行对象语义识别。由于外延是语义概念，因此可在外延中定义“键”以方便在外延范围内对实例进行唯一的语义识别。

(2) 通过键建立数据索引。由于索引建立在相关语义基础之上，具有了“键”就可以通过键值建立索引以提高查询效率。

2. 类与接口

继承和多态是面向对象技术中最具有特色的重要机制之一，也是其精华所在。从继承机制实现的技术角度考虑，ODMG 标准还将类型概念进一步细化，引入“类”和“接口”两个不同的概念。

(1) 接口(interface)。类型中只含有对象抽象行为（即操作）的称为接口，如果作为基类，其派生类只能继承所具有的操作。

(2) 类(class)。类型中同时含有对象操作与抽象状态（属性或联系）的称为类，类也可称为对象类型，如果作为基类，其派生类可以继承所具有的操作与状态。

接口和类的关联与区别如图 3-1 所示。

接口概念引入背景：当涉及的多个类具有相同方法但具有不同的对象范围（外延）时，可将其共同方法集中起来形成一个新的类型并看作接口，然后将涉及各个类作为其子类型完成接口继承。作为一种特定类型，接口具有可见的状态和方法，关键点在于其方法可被其他接口或类所继承，但其状态不能被继承下去。接口状态（属性或联系）只起到某种说明解释的作用，不具有作为其对象的状态变量的基本功能。因此，接口不可实例化，即不能通过接口直接创建实例对象。接口实际上相当于面向对象方法中的“抽象类”。

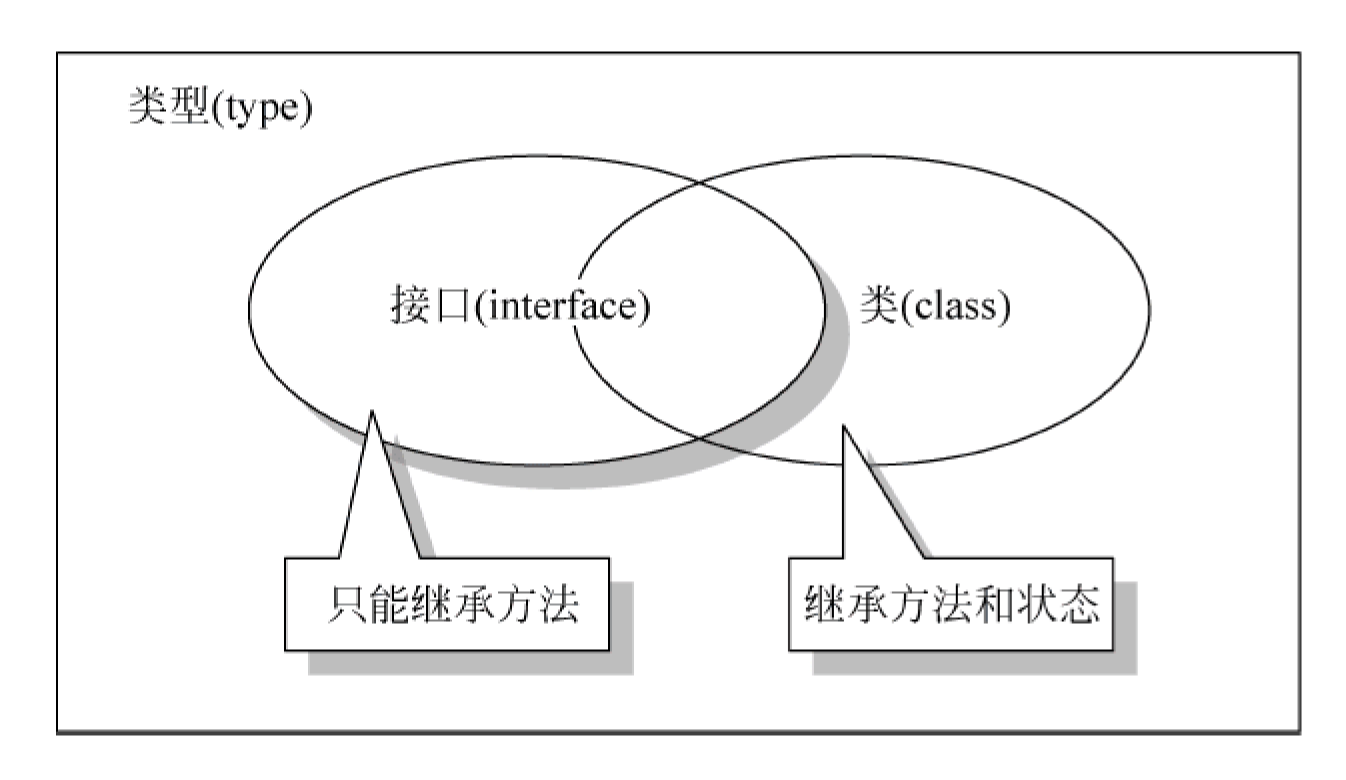


图 3-1　接口与类

接口外部说明是对象的抽象行为(方法)的详细描述,它指定了用来指定操作名、参数类型和返回值的操作签名。

一个接口声明的实例为:

```
interface ObjectFactory{
    Object new()
    };
```

说明:interface ObjectFactory 说明接口 ObjectFactory 的抽象行为,其中包含一个操作 new(),它返回一个带有对象标识 OID 的新对象。通过继承该接口,用户可以为每个自定义的原子对象类型创建对象实例。

3. 类的声明

类声明由一个外部说明(specification)和一个或多个实现(implementation)组成。其中外部说明主要定义类外部特征,包括类的状态、操作和异常,而实现主要是类的操作及其他各种细节的实现。类的外部说明与类的实现无关。

1) 类外延和键

RDB 需要将关系模式和关系实例区分开来,OODB 也需要将类定义和该类对象集合区分开来。前述的类外延就是一个类所有持久对象实例的特定集合,如同一个关系模式可以具有多个关系实例,一个类也可以通过类外延创建不同的对象实例集合。也就是说,当命名了一个外延,就相当于创建了一个对象实例集合。类外延创建格式为:

```
class 类名称(extent 外延名称)…
```

在 RDB 中,系统为每个关系模式维护一个或多个关系实例即关系外延;在 OODB 中,模式设计者可以选择系统是否需要为每个定义的类自动维护一个或多个对象实例集合即类外延。维护外延包括向外延集合中插入新创建的对象实例和从外延集合中移走已经删除的对象实例,或者建立并管理索引,提高外延集合中实例访问效率。管理索引将使系统开销增大,设计者需要单独说明是否要为外延建立索引。

如前所述,由于接口不能有相关对象实例集合,因此接口可以看作是没有(不能)定义外延的(抽象)类。实际上,接口主要应用于若干个类具有相同特性而具有不同外延时的情形,在某种意义上与关系模式所位于的相当层次类似,对于 RDB 而言,若干个不同的关系实例

可以具有相同的关系模式；对于 OODB 来说，不同的派生类可以从接口那里继承相同的操作。

不同于接口，类定义用于形成用户所需要的数据库模式和创建实际的应用对象。类可以实例化，其状态和方法都可以被子类型继承。因此，需要将数据库中已有的类定义与相应对象实例集合进行区别，这就和关系模式与关系实例间关系相似。在 ODMG 中，这种区别通过赋予类及其外延(extent)不同名称实现。因此，从创建类的语法结构上来看，类的名称就相当于类的设计模式，而类的外延就相当于界定该类当前存在的持久化对象实例集合。

在创建类的语法形式中，类外延声明方式是通过小括弧将关键词 extent 及紧跟其后的范围括起来，添加在类名称之后。通常将类名称声明为一个单数名词，而将相应类外延声明为该名词复数形式。

在某些情况下，类的对象实例可以通过实例本身的取值进行标识，这种可以标识对象的"对象取值"就是前面提及的对象的键(key)，这样就可在类外延中定义对象实例的键，用于在外延范围内唯一识别一个对象。键由一个或多个取值(属性或联系)组成，由一个取值组成的键称为简单键(simple key)，由多个取值组成的键称为复合键(compound key)。一个带有类外延的类可以拥有一个或多个键。键的创建格式为：

```
class 类名称(extent 外延名称,key(取值 1,取值 2,…,取值 n))…
```

【例 3-1】 具有外延和键的类声明如下。

```
Class Person
     (extent persons key social - number)
{attribute string social - number;
 attribute string name;
 attribute integer age;
};
```

2) 类声明

类声明主要由属性声明、联系声明和方法声明组成。

(1) 属性声明。属性(attributes)通过将固定数据类型的值与一个对象实例相关联，从选定角度描述对象实例的性质。需要注意的是，属性值只能是文字。属性取值可以是基本数据类型，也可以复杂数据类型。复杂数据类型主要有前述的由同类数据构成的聚集类型和由不同类数据构成的结构类型。

(2) 联系声明。联系(relationships)是指不同类的对象之间的对象的引用或对象聚集的引用。ODMG 通过联系描述同类型或不同类型文字、对象之间的关联。例如，大学和教师都可看作两个不同的类，那么，大学类中的一所大学(对象)可以关联教师类中的很多对象——教师，而教师类中的每个对象又都关联大学类中的一个对象(具体的大学)。这里就存在着两个联系：大学类到教师类联系 T1 和教师类到大学类的联系 T2，而 T1 和 T2 互逆，即 T1 和 T2 互为反联系，其中 T1 表达了大学与教师的 1∶n 联系。这与 E-R 模型中的二元联系类似。联系的参与者是对象，联系的值只能是 OID 或其集合。联系声明就是要描述这种类型之间的关联

(3) 方法声明。方法(methods)声明与 C++ 中函数声明相同，每个方法属于某个类，并且由该类的对象进行调用，因此类型中的对象可以看作是相应方法的隐含参数。在这种观

点之下，调用方法的对象决定具体调用方法，从而可以使不同类型中的对象使用相同的方法名，从而也就自然引入了重要的多态与重载(overloaded)机制。

在 ODMG 中，方法声明包括下述内容。

(1) 方法参数：可以分为输入、输出和输入输出 3 种类型，分别由 in、out 和 inout 表示。在实际应用中，in 参数通过"值"进行传递，而 out 和 inout 通过"引用"进行传递，因此，在方法中可以改变 out 和 inout 参数，但不能改变 in 参数。另外，方法还可以返回"值"。

(2) 异常处理：方法可能会引发异常(exception)，异常表示一种不正常或未曾预料到的情形，而异常由方法中的某个特定的方法进行处理，这种处理也可以是通过一系列的间接调用实现的。在 ODMG 中，可以在方法声明之后添加条件关键字 raise，再在其后添加由括号括起来一个或多个由该方法引起的异常描述。例如，假设一个方法表示两个实数相除，如果除数为零，则会引发"除数不能为零"异常报出。

3.3.4 接口继承与类继承

如前所述，由于数据库中的类能够通过语言联编映射为编程语言中的相应类，因此可以直接实例化；而接口不能直接实例化，因此，在 ODMG 中，"超类型——子类型"继承就需从技术层面上区分"接口"继承(ISA 关系)和"类继承"(EXTENDS 关系)两种情形。

在 ODMG 标准中，ISA 关系用于说明类型之间的行为继承，而 EXTENDS 关系用于说明类型之间的状态和行为继承。

1. 接口继承

接口主要用于声明可以被类或其他接口继承的抽象行为(操作)，因此，接口继承可以看作是"行为(操作)继承"，其创建格式为：

```
class 子类型(接口或类)名称:超类型(接口)名称{ … }
```

其中，{…}表示相应子类型特有的类型声明(下同)。需要注意的是，行为继承要求超类型必须是一个接口，而子类型可以是接口也可以是类。行为继承允许多重继承，即子类型可以从多个不同接口中继承行为。

2. 类继承

由于类概念中基本要素是状态和行为，对于状态和行为的继承就称为类继承。类继承的创建格式为：

```
class 子类型(类)名称 extends 超类型(类)名称{ … }
```

需要注意的是，在扩展继承中，超类型和子类型都必须是类，并且不允许进行多重继承。

图 3-2 所示为接口继承和扩展继承的直观描述。

下面的例 3-2 主要说明了接口继承与扩展继承的情形。

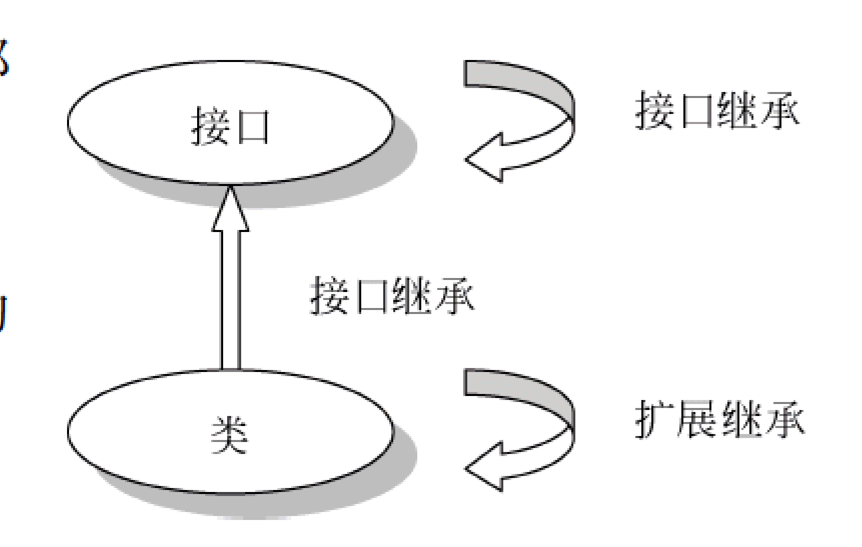


图 3-2 接口继承和扩展继承

【例 3-2】 接口继承与扩展继承实例。

```
Interface Employee{...};
Interface Professor:Employee{...};
```

```
Interface Associate_Professor:Professor {...};

Class Salaried_Employee:Employee{...};
Class Hourly_Employee:Employee{...};

Class Person{
    attribute string name;
    attribute date birthday;
};

class Employeeperson extends Person:Employee{
    attribute Date hireDate;
    attribute Currency payRate;
    relationship Manager boss inverse Manager::subordinates;
};

class ManagerPerson extends EmployeePerson:Manager{
    relationship set<Employee> subordinates inverse Employee::boss;
}
```

说明：假设在接口 Employee 中定义了操作 calculate-paycheck。上述接口 Professor 是对接口 Employee 的接口继承，而接口 Associate_Professor 又是对接口 Professor 的接口继承；类 Salaried_Employee 和 Hourly_Employee 也是对接口 Employee 的接口继承。类 EmployeePerson 是对类 Person 的扩展继承，类 ManagerPerson 又是对类 Employeeperson 的类继承。在接口继承中，还可以根据子类型的具体情况对接口 Employee 中操作 calculate-paycheck 进行修改。另外，也允许对行为进行多重继承，即一个子类型可以从不同的超类中继承多个具有相同名称、不同参数的操作。

3.4 ODMG 数据操作

在 ODMG 3.0 标准中，基于数据库技术框架的数据定义语言和数据查询语言分别称为对象定义语言（Object Definiton Language，ODL）和对象查询语言（Object Query Language，OQL）。ODL 类似于 SQL 中的 DDL，但具体写法有所不同，OQL 则具有一定的 SQL 风格。作为面向对象数据定义语言，ODL 主要用途在于创建对象类型，即类和接口声明。ODL 不是一种完全的编程语言，用户也可独立于任何编程语言而在 ODL 中制定一种数据库模式，然后使用特定的语言绑定来指明如何将 ODL 结构映射到特定编程语言中的结构，如 Smalltalk、C++ 和 Java 等。

3.4.1 对象定义语言

如同 C++ 中情形，OODB 中数据元素的创建包括类型创建和对象创建。

1. 类的创建

OODB 中的数据模式被定义为一系列接口和类（class）的集合。

1）接口声明

ODL 中接口声明（创建）包括下述内容。

(1) 关键字 interface 和接口名称。

(2) 用“：”表示接口继承。

(3) 用大括号“{}”包含的可以按照任意顺序出现的类型状态和方法列表。

2) 类声明

ODL 中类声明(创建)包括下述内容。

(1) 关键字 class 和类名称。

(2) 用关键词 extends 表示类继承。

(3) 类外延 extent 和外延名称。

(4) 用大括号“{}”包含的可以按照任意顺序出现的类型状态和方法列表。

【例 3-3】 创建一个广东省内大学的面向对象数据库模式,该模式包含有类“Course”“Person”“Student”和“Department”等,其中假设已经存在对象工厂 ObjectFactory。

① 类 Course 声明。

```
class Course: Object(extent courses,key course_number){
          /* Object 是 Course 的超类型,以下为类的公共性质,其取值不以实例而异。
             外延名称为 courses, course-number 是键,以下为类的性质,其值以实例而异,是类
             型的非公共性质。 */
    attribute String course_number;              /* 课程号,字符串型 */
    attribute String name;                       /* 课程名称,字符串型 */
    attribute Integer credit;                    /* 学分数,整数型 */
    attribute Enum{spring,fall} time_offered;    /* 开课时间,枚举型,春季或秋季 */
    relationship Set<Course> has_prerequisites inverse Course::is_prerequisite_for;
          /* has_prerequisites 表示本例有哪些先修课程,is_prerequisite_for 表示本例是哪些
             先修课程,这是 Course 中两个互逆遍历路径。 */
    relationship Set<Student> taken_by inverse Student::take;
          /* taken_by 是个遍历路径,通过它可以查询本门课程的学生。在 Student 有遍历路径
             take。以下是例的操作 */
    offer(in Enum{spring,fall })raises<already_offered>;
          /* 将本例中 time_offered 属性为输入参数 in 所指定的学期,Enum{spring,fall}为输入
             参数的类型。如果已经规定了开课时间,则此操作是重复设置,会引起"already_
             offered"异常情况 */
    drop()raises(not_offered);
          /* 清除本例中 time_offered 属性,表示暂时不开此课。若已不开此课程,则没有必要执
             行此操作,否则会引起"not_offered"异常情况 */
};
```

② 类 Person 声明。

```
class Person: Object{
     attribute String name;
     attribute String birthday;
     attribute String sex;
     age(in Person, birthday);          /* 以 Person、birthday 为输入参数,计算当前年龄 */
};
```

③ 类 Student 声明。

```
class Student extends Person(extent students key student_number){
```

```
        /* 其他属性从超类型 Person 继承 */
    attribute String student_number;
    relationship Department departmentinverse Department::student;
        /* department 是遍历路径,表示学生所在的系,在 Department 类型中,有遍历路径 student,
           表示有哪些学生 */
    relationship Set<Course> take inverse Course::taken_by;
        /* 通过 take 遍历路径,可以查询本实例所选课程,Course 中的 taken_by 是其逆遍历路径 */
    Transfer_department(in Department) raises(department_full);
        /* 本例转系,in 是输入参数,表示要转入的系.如果该系学生已经满员,则引起"department_
           full"异常情况 */
};
```

④ 类 Department 声明。

```
class Department:Object(extent department key department_number){
    attribute String department_name;    /* 系名称,字符串型,键 */
    attribute integer number_of_faculty;/* 教师人数,整数型 */
    relationship Set(Student)student inverse Student::Department;
    add_a_student(in Student);           /* 增加一个学生,in 为输入参数,表示新增的学生 */
    drop_a_student(in Student);          /* 取消一个学生,in 为输入参数,表示去掉的学生 */
};
```

说明:类的状态(静态特性)包括属性和联系,通过"属性"可描述类的基本数据信息,但在许多情况下,类的主要关联信息可以通过该类对象与其他相同或不同类中对象之间的"联系"表现出来。此时,类之间对象的双向联系是 OODB 中类定义的一个显著特色。

上述①在类 Course 和类 Student 之间定义联系为:

```
relationship Set<Student> taken_by inverse Student::take;
```

这里,relationship 是关键字,taken_by 是联系名,Set<Student>是相联系的类 Student 中的对象集合。事实上,如果将 taken_by 看作一个属性,其属性值就是 Set<Student>,即 Course 中每一个对象都和类 Student 中一组对象相连接。

另外,类 Student 中的每一个对象也会和类 Course 中的一组对象联系,其联系名称为 take。这样类 Course 和类 Student 中对象就存在着一种"互逆"联系。联系语句短语"inverse Student::take"就表明这种"互逆"语义,即类 Student 中联系 take 是类 Course 中联系 taken_by 的"逆联系"。由于联系 take 在另一个类 Student 中定义,因此联系名称就加上了该类名称 Student 和符号::,这是由于通常引用定义在其他类中某些事物(如特征和类姓名等)时通常都适用的符号::。

这样,上述整个联系语句语义表明,类 Course 中每个对象(course)都联系类 Student 中一组对象(student),而且这组 student 中的每一个也都通过类 Stutent 中联系 take 与类 Course 中一组对象(course)联系。

上述联系是建立在不同类 Course 和 Student 之间。而"①"中还将类 Course 在逻辑上看作两个"不同"的类建立了下述联系语句,即在"同一类"Course 中建立联系,这相当于 RDB 中的关系自连接。

```
"relationship Set<Course> has_prerequisites inverse Course::is_prerequisite_for;
relationship Set<Course> is_prerequisite_for inverse Course::has_prerequisites;"
```

其具体语义可以仿前述予以解释。

由上述可知，对两个类 C1 和 C2 中联系名称分别为 T1 和 T2 而言，相应联系语句格式可以表示为：

在类 C1 中

```
relationship Set < C2 > T1 inverse C2::T2;
```

在类 C2 中：

```
relationship Set < C1 > T2 inverse C1::T1;
```

如果 C1＝C2＝C，则相应语句还可以省略逆联系中的类 C 而简写为：

```
relationship Set < C> T1 inverse T2;
relationship Set < C> T2 inverse T1;
```

对于②、③、④中的联系语句也可做相同的语义解释。

2. 对象创建

如前所述，OODB 中的对象分为原子对象和非原子对象两种情形，而非原子对象在应用中多表现为聚集对象。

原子对象是具有完整“对象三要素”的用户自定义对象，可以呈现出各种不同的复杂情形，需要用户通过接口或类定义等完成创建过程。

聚集对象相对比较整齐，根据 ODMG 标准，通常是使用为不同聚集对象定义的一种特殊接口——对象工厂接口中定义的构造器函数来创建聚集对象。

【例 3-4】 如前所述，下述语句定义了 ODMG 标准中一个对象工厂接口。

```
Interface ObjectFactory{
Object new();
}
```

通过该工厂对象接口，可为用户创建聚集对象为：

```
interface DateFactory:ObjectFactory{
    exception InvalidDate{};
    …
    Date calendar - date(in unsigned short year,
    In unsigned short month,
    In unsigned short day)
    raises(InvalidDate);
    …
    Date current();
};
```

说明：上述工厂接口 ObjectFactory 只有一个操作 new()，它返回一个带有对象标识 OID 的新对象。新创建的 DateFactory 接口继承该接口生成函数 new()，就可为每个用户定义的原子对象创建它们自己的工厂接口，并且可以针对每种新对象实现不同的 new()操作。例如，给定一个 DateFactory 接口，它还可以有其他一些操作，如 calendar-date 用来创建一个新的日期，current 用来创建一个值为当前日期的对象。

3.4.2 数据查询语言

作为支持 ODMG 对象类型查询语言，OQL 具有下述基本特征。

(1) 结合编程语言设计。OQL 基本思路是与面向对象程序语言密切配合共同使用，OQL 和一个 ODMG 绑定的编程语言(如 C++、Java 等)整合，这样，嵌入某种编程语言的一个 OQL 查询就可返回与相应语言类型系统匹配的对象。

(2) 具有 SQL 风格。OQL 查询语法与关系型 SQL 语法相类似，采用 SQL 风格，同时增加了有关 ODMG 的基本概念特征，如对象标识、复杂对象操作、继承多态和联系等。但 OQL 与 SQL 只是风格相似，本质上并不兼容。由于上述特征“(1)”，OQL 可单独使用，也可嵌入到相应面向对象程序设计语言当中。

(3) 行为包含数据更新。OQL 不提供显式数据库中数据的更新操作，如插入、删除和修改等，这主要是基于封装和安全性的考量，这些常规数据操作主要由类定义中的相应方法实现，也就是说，ODMG 模式中数据操作是通过类型行为定义中面向对象编程语言编写相应代码实现，而不是由系统统一提供。

1. 基本查询语句

对象查询语言 OQL 是以面向对象模型为基础的类 SQL 查询语言，允许使用具有传统 SQL 风格的 SELCET 语句完成书写表达。在 collection 和 structure 中，如果数据是由类型组成的，则用“< >”表示，如果是由文字组成的，则用“()”表示。

通常，OQL 中一个 SELECT 语句包含下述基本内容。

(1) SELECT 子句：关键字 SELECT＋表达式列表。

(2) FROM 子句：关键字 FROM＋一个或多个变量声明。

通常变量声明格式为：值为聚集的表达式＋AS(可选)＋变量名称。

需注意的是，SELECT 语句本身就是一个聚集值表达式，FROM 子句中可再嵌入 SECLECT 语句。

(3) WHERE 子句：关键字 WHERE＋布尔表达式。

如同传统 SQL，布尔表达式只能使用常量或在 FROM 子句中定义的变量。表达式中的不等号用< >表示，各种比较操作符和逻辑运算符都与传统 SQL 相同。

【例 3-5】 在例 3-3 数据库中，用 OQL 的 SELECT 语句完成如下查询。

查询大学中授课门数超过两门的教师，并要求显示学校名称和教师姓名，显示时属性的别名分别为 university_uname 和 faculty_name。

```
SELECT university_name:F. works_for.uname,faculty.name:F.name
FROM faculties F
WHERE F.num_teach()> 2;
```

【例 3-6】 查询广州地区大学教师开设课程名。

```
SELECT DISTINCT C.cname
FROM universities U,U.staff F,F.teach C
WHERE U.city = 'guangzhou';
```

说明：关键字 DISTINCT 消除结果中重复部分，查询结果为集合< set >，如果包含有重复数据则为包< bag >。

【例 3-7】 上述查询也可以用子查询表示，但查询出现在 FROM 中。

```
SELECT DISTINCT C.cname
FROM (SELECT *
      FROM universities U
      WHERE U.city = 'guangzhou')D1,
     (SELECT F
      FROM D1.staff F)D2,
      D2.teach C;
```

说明：本例语句实际上没有上一个查询语句简洁，但说明 OQL 中可建立起新的查询形式。这里 FROM 子句具有 3 个嵌套循环。在第一个循环中，变量 D1 覆盖了广州地区所有大学，这是 FROM 子句中的第一个查询的结果。对于嵌套在第一个循环外的第二个循环，变量 D2 覆盖了大学 D1 的所有教师。对于第三个循环，变量 C 覆盖了该教师的所有任课，而此语句不需要 WHERE 子句。

【例 3-8】 上例中也可用在 WHERE 子句中嵌有子查询的形式。

```
SELECT DISTINCT C.cname
FROM coursetexts C
WHERE C.teacher IN
        (SELECT *
         FROM faculties F
         WHERE F.works_for IN
              (SELECT U
               FROM universities U
               WHERE U.city = 'guangzhou'));
```

【例 3-9】 查询中山大学教师，要求按年龄降序排列，若年龄相同按工资降序排列。

```
SELECT *
FROM universities U,U.staff F
WHERE U.uname = 'San Yet Sen university'
ORDER BY F.age DESC,F.salary DESC;
```

说明：通常 SELECT 语句中查询结果为集合(set)或包(bag)，但加上 ORDER BY 后，输出结构就为列表(list)。在集合或包中，可不考虑输出行的顺序，但在列表中，需要考虑相应的行序。

【例 3-10】 以下查询返回为列表而不是集合或包。

```
(SELECT F.fno,F.name
 FROM faulties F
 ORDER BY F.age DESC)[0:4];
```

说明：表达式[0：4]表示抽取年龄最大的 5 名教师，这相当于 RDB 中的 TOP 子查询。

【例 3-11】 查询广州地区各大学中教师开课的课程名称，要求显示校名、教师名和课程名。

```
SELECT Struct(U.uname,set(F.uname,set(C.cname)))
FROM universities U,U.staff F,F.teach C
WHERE U.city = 'guangzhou';
```

说明：SELECT 子句中表达式可以是简单变量，也可以是任何表达式，包括类型构造符构成的表达式。本查询就用了 struct 类型构造符合 set 类型构造符。SELECT 中的 struct 是一种显式定义结构类型的方式，在实际使用中可以省略。

2. 查询表达式

如前所述，OQL 还可以使用表达式来完成查询，查询结果就是表达式的取值。同时，表达式还可以用来构成其他 OQL 语句。表达式可以是变量、文字、命名对象和查询语句，以及它们通过聚集(平均、计数、求和、求最小值和最大值及分组)、逻辑等运算(使用含有全称量词 FOR、ALL 和存在量词 EXISTS 表达式)和集合运算(并集运算、交集运算和差集运算)得到的复合运算表达式。

1) 聚集表达式

与 SQL 相同，OQL 也有 5 种聚集操作：AVG、COUNT、SUM、MIN 和 MAX。但在 SQL 中，聚集运算只作用在关系表的指定列，而在 OQL 中可作用于所有含有适当类型的集合上。其中，COUNT 可用在任意聚集类型，SUM 和 AVG 可用在基本类型(整型等)的聚集，MAX 和 MIN 可用在任何可比较类型(整型和字符型)的聚集上。

【例 3-12】 在例 3-3 数据库中，查询每个年龄段教师平均授课门数。

```
SELECT F.age,
avgNum:AVG(SELECT P.F.num_teach() FROM partition P)
FROM faculties F
GROUP BY F.age
```

说明：对教师按年龄段分组，每一个分组用关键字 Partition 表示，这样在 SELECT 子句，就可以对每一个分组操作了。此处统计教师授课门数利用了函数 num-teach。

【例 3-13】 查询以 40 岁为界的两个年龄段的教师平均授课门数。

```
SELECT low,high
    avgNum:AVG(
        SELECT P.F.num - teach()
        FROM partition P
        )
FROM faculties F
GROUP BY low:F.age < 40,high:F.age > = 40;
```

说明：本查询中，分组子句只产生两个分组，称为 low 分组(age<40)和 high 分组(age≥40)，faculties 中教师根据年龄值放在两个分组中。在 SELECT 子句中，low 和 high 是两个布尔变量，在输出的每个元组中，只有一个是 true 值。这个查询只输出两个元组，其中一个元组的 low 值是 true，avgNum 值是小于 40 岁教师的平均授课门数；另一个元组的 high 值是 true，avgNum 值是大于或等于 40 岁教师的平均授课门数。

【例 3-14】 查询至少有一位教师年龄超过 70 的大学编号、校名和教师人数。

```
SELECT U.uno,U.uname,U.num - staff()
FROM universities U
```

```
GROUP BY U.uno,U.uname
    HAVING MAX(
          SELECT F.age
          FROM patition P,P.staff F
      )>70;
```

说明：上述查询根据大学分组，在 HAVING 子句中，挑选至少一位教师年龄超过 70 岁(即教师中最大年龄超过 70 岁)的那些组，然后再求每组中有多少名教师。

2）量词表达式

量词表达式用于检测一个集合所有成员或至少存在一个成员是否满足给定条件。

① 所有成员满足给定条件检测。OQL 通过表达式 FOR ALL x IN S：C(x)方式使用任意量词。该表达式用于检测集合 S 所有成员 x 是否满足条件 C(x)。如果满足，其结果就是 true，否则就是 false。

② 存在成员满足给定条件的检测。OQL 通过表达式 EXISTS x IN S：C(x)方式使用存在量词。该表达式表示如果 S 中至少有一个成员 x 满足 C(x)，结果就为 true，否则就为 false。

【例 3-15】 在例 3-3 中，查询存在 60 岁以上教师的大学名称。

```
SELECT DISTINCT U.uname
FROM universities U
WHERE EXISTS F IN U.staff:F.age>=60;
    /*F是元组变量,谓词C(x)为F.age>=60*/
```

【例 3-16】 查询教师年龄全在 50 岁以下的大学名称。

```
SELECT DISTINCT U.uname
FROM universities U
WHERE FOR ALL F IN U.staff:F.age<50;
```

3）集合运算表达式

【例 3-17】 在例 3-3 中，查询教师人数不超过 1000 人，但工资低于 1500 元的人数超过 500 人的那些大学的编号和校名，INTERSECT 集合运算符表示交集。

```
(SELECT U.uno,U.uname
  FROM universities U
  GROUP BY U.uno,U.uname
      HAVING U.num_staff()<1000)
INTERSECT
(SELECT U.uno,U.uname
  FROM uneiveisities U,U.staff F
  WHERE F.salary<1500
  GROUP BY U.uno,U.uname
      HAVING U.uno_staff()>500);
```

4）路径表达式

路径表达式是 OQL 表达式的关键部分，其作用可使给定查询深入到复杂属性(联系)和作用于对象之上的方法，同时也可简化查询表示。类似于 C++中的情形，OQL 通常使用点符号“.”访问对象或结构的相关成分。设 O 表示类 C 中某个对象，P 表示类 C 中某个要

素(属性、联系或方法),O. P 表示 P 作用于 O 上的结果,则

(1) 如果 P 表示一个属性,则 O. P 表示对象 O 在 P 上属性取值。

(2) 如果 p 表示一个联系,则 O. P 表示通过 P 引用与对象 O 相联系的对象集合。

(3) 如果 P 表示一个方法,则 O. P 表示方法 P 作用于对象 O 的结果。

在例 3-3 中,设 student 是类 Student 中的对象实例,则有 Student. age 表示对象 Student 在属性 age 上取值,即对象 Student 的年龄;Student. take 表示对象 Student 通过联系 take 与类 Course 中相关联的一组对象。

3. 宿主语言变量赋值

嵌入式 SQL 需要在元组分量和宿主语言变量之间传递数据,在 OQL 中,可将查询表达式结果自然地赋值给任何适当类型的宿主语言变量。

【例 3-18】 查询年龄大于 60 岁教师。

```
SELECT *
FROM faculties F
WHERE F.age > 60;
```

说明: 查询结果是年龄大于 60 岁的教工集合,其数据类型是 set(faculty)。如果 oldfaculties 是同类型的宿主语言变量,则可使用经过 OQL 扩充的 C++ 写成如下形式。

```
oldfaculties = SELECT *
               FROM faculties F
               WHERE F.age > 60;
```

此时,oldfaculties 的值就是这些 faculties 对象的集合。

4. 聚集中提取元素

SELECT 语句查询结果通常都是聚集(集合、包或列表),实际应用可能需要从中提取某个或某些元素。

① 当聚集中只含有一个元素时,为单元素聚集。OQL 提供操作符 ELEMEMT 将单元素聚集转换为单个元素。

② 当聚集中含有多个元素时,首先通过 ORDER BY 子句将集合或包转化为列表 L,然后对于获得的列表使用序号访问其中所需的元素,从而实现从聚集中提取元素。列表 L 的第 i 个元素可以用 L[$i-1$]得到,因为列表序号是从 0 开始的。

【例 3-19】 查询大于 60 岁的教师,查询结果为按照工资、年龄降序排列的列表形式。

```
facultyList = SELECT *
              FROM faculties F
              WHERE F.age > 60
              ORDER BY F.salary DESC, F.age DESC;
```

说明: 此语句将工资、年龄降序排列的所有 faculty 对象的列表赋予宿主语言变量 facultyList,然后通过元素序号访问所需元素。对比 RDB 中的游标访问多行数据的技术,这种方法将更为自然、简单和方便。

【例 3-20】 按照工资和年龄降序排列打印年龄超过 60 岁的教师属性(工资、年龄、姓名和工号)的 C++ 函数为:

```
facultyList = SELECT *
FROM faculties F
                WHERE F.age > 60
                ORDER BY F.salary DESC,F.age DESC;
     /*对 faculties 进行排序,得到结果为方法变量 facultyList,类型是 List(faculty)。*/
number()fFaculty = COUNT(facultyList):
     /*用 OQL 运算符 COUNT 计算教师的数量。*/
     /*以下直到结束是 for 循环,在该循环中整数变量 i 覆盖了该列表的每个位置。为了方便,把
       列表的第 i 个元素赋给变量 faculty,然后在第 1 行和第 n 行再打印教师的相关属性*/
for(i = 0:1 < numberOfFaculty:i++)
   {faculty = facultyList[i]
          cout << faculty.salary <<""<< faculty.age <<""
            << faculty.name <<""<< faculty.fno <<"/n";
    };
```

说明:上述语句中的""是将 faculties 转换为列表形式,并将列表结果赋予宿主语言变量 facultylist,然后通过循环语句进行逐个提取列表中元素。

本章小结

RDB 自其诞生以来一直都是商用数据库的主流和霸主,并在可以预见的将来,这种地位似乎也不易动摇。但作为一种技术,RDB 却难以包打天下,它有着自身的应用边界。RDB 对于涉及事务处理的业务范围十分有效,但对于一些发展迅猛的新兴领域,如网络数据、多媒体数据和移动对象数据等,其不足与弱势就显露出来。这些弱势包括数据类型简单、结构与行为分离、查询实现复杂和阻抗失配等,而这来源于关系数据模型的自身限制,不能仅仅采用简单的补救。由此,对象数据库应运而生,成为新一代(第三代数据库)的典型代表之一。

对象数据库分为两种类型。其一是面向对象程序设计语言的持久性扩充,这就是本章讨论的面向对象数据库 OODB;其二是关系数据库的对象扩充,这就是第 4 章将要讨论的对象关系数据库 ORDB。

OODM 从本质上来看,实际上是将面向对象方法中的基本元语和重要机制在数据库的数据模型框架中进行适当的配置和解释,这样做的初衷自然是想借用面向对象方法所具有的强大语义表达能力有效处理机制来克服常规数据库的缺陷与不足。但由于程序语言中操纵的数据具有瞬时性,与数据库需要存储的持久性数据难以适配,因此,面向对象程序语言的持久化就是 OODB 所需要解决的首要问题。一个简单的方式是将所涉及类中的所有对象都进行持久化,也就是"按类持久",但这会带来持久对象管理开销过大的问题,而在实际应用中,一个类中并不是所有对象都需要进行持久化处理的,因此,对象持久化是一个需要精细分析解决的问题。

在解决对象持久化之后,面向对象中的方法技术还不能直接照搬到 OODB,还需要按照数据库管理技术指向对其中元语和机制进行数据库技术框架下的语义解释和业务处理工作。这种释义过程的落脚点就是 OODB 的技术实现,而这种规范化的释义就是 ODMG 中的 C++持久化标准。

在 ODMG 标准中,按照是否同时具有状态和方法而将 C++的"对象"具体区分为"原子

对象”和“构造对象”；按照是否具有“附加”对象标识而将 C++ 中对象具体区分为“对象”和“文字”。此外，从抽象层面的方法描述和技术层面上的方法实现角度将“类型”和“类”进行必要区分；从方法继承角度将“类”和“接口”进行了必要区分，进而将 C++ 中的继承机制确定为“类继承”和“接口继承”两种情形。此外，如果将类的定义看作一种数据模式，则在数据库的实际应用中一种数据模式需要对应多个数据实例，如关系模式与关系实例的对应。此时，就需要对同一类模式对应的不同对象实例集合进行界定和描述，也就引入了类外延的概念。类外延实际上就是相应数据集合的域约束，这对于数据管理和数据查询都具有十分基本的意义。所有这些都可以看作是 ODMG 标准对于基本元语和重要机制进行的关于数据库语义方面的进一步解释，同时也是作为对象数据库实现的更为明确和实用的技术指向。这种数据库语义释义对于将数据库技术与各类新兴计算机技术有效整合提供了良好的示范效用和研究借鉴。

ODMG 数据语言主要由对象数据定义语言 ODL 和对象数据查询语言 OQL 组成。ODL 不是一套独立的编程语言，用户可以相当灵活地使用；OQL 则具有明显的 SQL 风格，用户可以对照 SQL 进行学习使用。

主要参考文献

[1] Abraham Silberschatz，Henry F Korth，Sudarshan S. 数据库系统概念[M]. 5 版. 杨冬青，唐世渭，译. 北京：机械工业出版社，2007.

[2] 王意洁. 面向对象的数据库技术[M]. 北京：电子工业出版社，2003.

[3] 徐洁磐. 面向对象数据库系统及其应用[M]. 北京：科学出版社，2003.

[4] 王能斌. 数据库系统教程(上册)[M]. 2 版. 北京：电子工业出版社，2010.

[5] 何新贵，等. 特种数据库技术[M]. 北京：科学出版社，2000.

[6] Matt Weisfeld. 面向对象的思考过程[M]. 杨会珍，尹清辽，等译. 北京：中国水利水电出版社，2004.

第 4 章 对象关系数据库

面向对象数据库自 20 世纪 90 年代已成为数据库技术的研究热点，人们对其寄予厚望并曾预言：作为第三代数据库主流技术，面向对象数据库将会取代传统关系数据库。然而其后形势并没有向人们所预料的方向发展。相反，面向对象数据库系统推出后，市场占有份额一直不大，根本无法与关系型系统抗衡。不要说“取代”，甚至连像样的“威胁”都难以企及。究其缘由，主要有下述几个原因。

(1) 从实际应用市场来看：在社会经济活动中，数据库最大的应用领域依然是传统的企事业单位事务性数据管理。在此领域，有关处理复杂对象、扩充数据类型和设计相应特定函数等并非主流需求，但却存在大量的需要进行联想式访问的查询需求，即任务攸关(mission critical)查询。这是关系数据库强项，面向对象数据库却难以胜任。

(2) 从数据库共享性来看：面向对象数据库主要处理 CAD 和 CIM 等复杂数据对象，而这些对象并不适合多用户的数据共享和频繁的事务联机处理。同时由于用户自定义数据操作的加入，使得面向对象系统的安全性面临更多、更困难的挑战，对于系统的故障恢复也有如此问题。对于大多数企事业单位而言，与增加数据库某些新的功能相比较，数据库的安全性和故障恢复则更为基本和重要。

(3) 从系统替代开销来看：将已经运行的关系型系统更换为面向对象系统是一个相当困难且痛苦的过程。这需要重新设计数据模型、转换存储数据、对新系统进行正确性验证和修改调试原有应用程序。由于面向对象系统的数据语言并未与 SQL 兼容，这样的替换工作会严重影响甚至中断一个单位的业务活动。另外人员的重新培训也增加了用户投资，降低了更换过程中系统的业务质量。

由于上述原因，人们在数据库“对象化”方面另外开辟了一条道路，即在关系型系统上增加面向对象某些常用的基本功能，这种具有对象扩充功能的新型数据库系统就是对象关系数据库系统(Object Relational Data Bases，ORDB)。ORDB 系统本质上还是关系数据库(Relational Data Bases，RDB)系统，其特征是在 RDB 上通过引入复杂数据类型突破 1NF 限制，通过引入“继承”和“引用”等技术增添面向对象过程部分功能。由于 RDB 主要用于广泛的事务处理，并不需要使用面向对象系统的所有功能，因此对 RDB 做一些必要的面向对象扩充通常也可满足大部分的实际应用需求。另外，OODB 并没有成熟的商业化产品，通过吸取面向对象技术基本概念，可让面向对象方法搭载在 RDB 平台之上先行进入市场，也不失为一种明智选择。其后形势发展也证明确实如此。特别是 SQL 已经被人们广泛接受——正如人们所说，“SQL 语言不管其好坏，仍然是星系间数据对话的语言”——ORDB 作为 RDB 的面向对象扩展，其数据操纵语句自然也是常规 SQL 的扩展。离开 SQL 风格的数据库语言，从当今态势来看，确实难以打开市场。正是在这种背景之下，ORDB 才得到人

们的广泛关注，并且迅速进入市场。

本章主要介绍 ORDB 基本技术。

4.1 数据与数据查询

数据查询是数据库最基本的技术支撑功能，不同的实际应用产生不同复杂程度的各类数据，而相应地也就有不同复杂层面上的数据查询技术。当然，“简单”与“复杂”的界定依赖于相应的应用背景和领域特征，同时也是相对的，需要结合学科技术要求进行具体分析。

4.1.1 数据管理分类矩阵

数据是数据库的基本要素，而持久存储和管理数据是数据库的核心功能。从 DBMS 技术实现的角度来看，可以将数据分为“简单数据”与“复杂数据”两种类型，同时将数据库基本功能分为“简单查询”和“复杂查询”两种情形。

1）简单与复杂数据

从数据管理应用现状而言，具有较高抽象层面数据模型的数据相对而言可以看作是简单类型数据，而具有更多语义特征的数据模型可以看作是复杂数据。

例如，关系数据模型具有较为简洁的数据类型和较高的抽象级别，可以看作是数学中笛卡儿乘积及其相关要素在数据库中的技术性释义，相对而言关系型数据就属于“简单类型”数据；而面向对象数据模型具有更为复杂的数据类型和更为丰富的语义特征，是面向对象思想与技术的数据库实现，同样相对对象型数据而言就属于“复杂类型”数据。

2）简单与复杂查询

如果数据库系统数据管理工作主要围绕数据查询展开，同时管理所有能够实现系统级别上统一的和完整的查询实现机制，则可以认为是复杂查询；如果数据查询只是整个数据管理中的一个环节，系统提供的非查询功能在整个系统机制中占有更大比例，则就可以认为是简单查询。

对于 RDB 而言，数据查询是数据库功能机制的主体，通过“投影”“选择”和“连接”3 种基本查询技术能够实现“视图”“多表整合”和“查询优化”等高层次的完整查询要求，因此关系数据查询可以看作是复杂查询；OODB 由于其技术特征限定，没有提供统一完整的系统级别查询机制，同时由于其着眼于 CAD 等复杂类型数据管理，数据查询不是其频发任务，更多的是通过应用程序对存储的数据对象进行非查询处理，因此从查询角度而言，可以看作是简单查询。

从这个观点出发，人们通过由“简单数据”“复杂数据”的数据维度和“简单查询”“复杂查询”的查询维度为相应的数据管理技术分类建立“DBMS 分类矩阵”，如图 4-1 所示。

4.1.2 基于分类矩阵的数据管理系统

在图 4-1 中，横轴为数据类型轴，纵轴为数据查询轴，两坐标轴都具有“简单”与“复杂”两个标度。如前所述，对于任何一个有效管理数据的 DBMS 都可以放入图中 4 个区域当中的某一个区域进行分类识别。

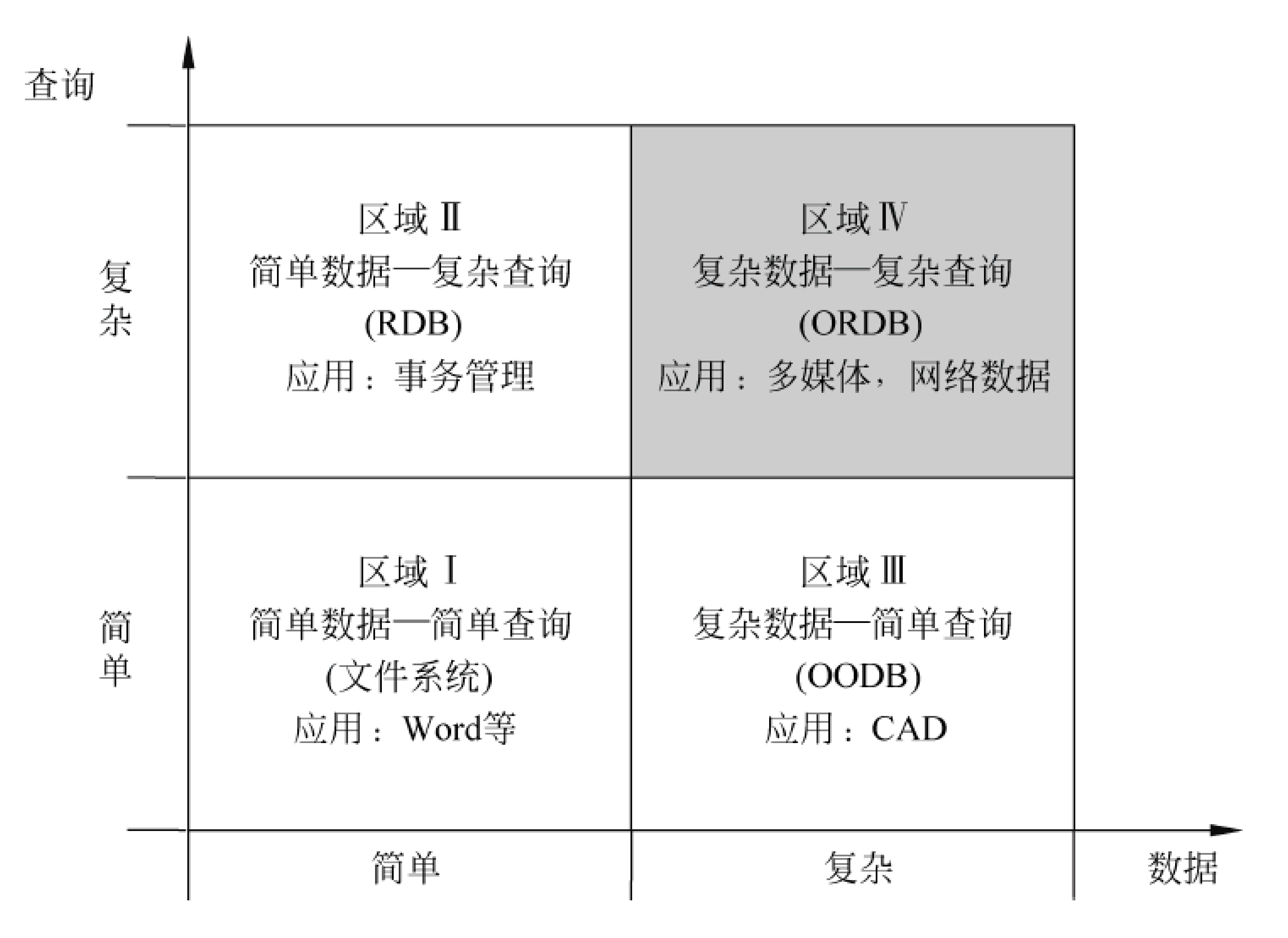


图 4-1　DBMS 分类矩阵

1）文件系统位于区域Ⅰ

区域Ⅰ的特征是"简单数据—简单查询"。

此时数据多为基于字符的文本文件，基本操作主要是读/写文件，也就是说，数据查询在整个数据操作过程中并不占有主导地位，即使需要查询，也只是借助操作系统而进行遍历式查找。这就是如今广为使用的各类文件系统，如 Office 中的各类基本组件等。

2）RDB 系统位于区域Ⅱ

区域Ⅱ的特征是"简单数据—复杂查询"。

实际应用中的企业事务管理和各类常规信息管理系统都位于这个区域，其中的后台支撑就是 RDB。RDB 管理的数据元素主要是元组和元组集合——关系表，数据结构相对简单，但其所有数据管理都是围绕数据查询展开的，并提供从数据创建、查询优化、视图管理、并发访问和故障恢复在内的一整套系统级别的数据查询机制。其中的非过程化数据查询语言更是其他各类数据库数据查询语言的典范和附着点，因此属于数据复杂查询范畴。以 RDB 为代表的"简单数据—复杂查询"类型是目前最大的一类数据库应用领域，也是当今难以替代的数据库商业热点。

3）OODB 系统位于区域Ⅲ

区域Ⅲ的特征是"复杂数据—简单查询"。

位于此区域的数据库应用主要是包括计算机辅助设计（CAD）、计算机辅助制造（CAM）、计算机辅助工程（CAE）、机器人学（Robotics）和计算机辅助生产管理（CAPM）等在内的计算机集成制造系统（CIMS），其基本点是研发适合于 CAD 应用的 DBMS。以 CAD 为例，其数据管理过程通常是将芯片的某种设计方案作为复杂数据持久存储在数据库当中。在需要处理时，先将其由外存读入内存，再使用已有相关计算机程序对其进行修改优化，最后将优化处理后的设计方案存回到数据库当中。在相关数据管理过程中，通常并不需要进行频繁的数据查询，甚至不需要查询操作，主要是实现设计方案的读取操作、计算操作与写回操作。这在本质上与文件系统的数据管理过程类似，数据管理的主体是"读"和"写"，只是

由于数据相当复杂,以人工操作为特征的文件系统难以有效地读入相关信息以将数据由外存格式转换为内存格式,在计算操作完成后同样难以承担相应的“逆转换”工作。另外,这类数据由于其逻辑描述和存储格式的复杂性,RDB 也难以胜任相关工作。实际上,第 3 章所讨论的 OODB 的主要应用背景就是 CAD。OODB 以 C++ 为基础,引入持久性对象存储 CAD 数据,通过增加数据库共享机制,将复杂对象的持久性存储管理和相关计算机处理密切整合,可以看作是区域Ⅲ的主要代表。

4) ORDB 系统位于区域Ⅳ

区域Ⅳ的特征是“复杂数据—复杂查询”。

由于计算机应用领域的日益拓展和数据库技术的普及发展,原先被认为是难以使用数据库的部门和领域都在积极研讨和推进数据管理技术的应用,其中的显著代表就是多媒体数据领域和网络数据领域。这两个领域的数据对象都具有较高的复杂层次,相关数据管理也都需要围绕基于内容的频繁数据查询而展开。由于数据的复杂性,因此需要建立统一的外存—内存相互转换机制;由于复杂数据操作中数据查询的主导性,因此除了系统级别的数据操作机制外,还需要对于数据对象个别建立相应的计算机程序操作。这就需要在现有数据库技术的基础上与面向对象程序语言进行有效适配整合,其中基本点就是扩展现有数据库(如 RDB 的基本数据(类型)到复杂数据(类型)),同时能够在复杂数据对象层级上通过函数实现相应计算机操作。本章讨论的 ORDB 正是在这样的逻辑框架和应用背景下产生和逐步发展起来的。

4.2 对象关系数据类型

RDM 在逻辑上具有较高的抽象层次,数据结构相当简洁,只有一个核心概念(即关系表),在应用中适合于常规的数据事务处理,如银行业务、票务办理、酒店预订、工资和人事管理等。随着计算机技术的发展,数据库进入到一个更为广阔的应用领域,需要对多媒体数据、网络数据和移动对象数据等进行有效管理。面对新的应用需求,RDB 显现出自身的一些局限性。

(1) 不支持复杂数据类型。复杂数据类型正是新型数据管理应用中的基本对象。

(2) 限制高级程序语言直接访问。复杂数据的处理需要高级程序设计语言的参与,RDB 自身特征的限制使得(如 C++或 Java 等)编写的程序难以有效访问数据库数据。

由于具有较强的语义表达能力和提供更好的程序组织形式及程序可靠性,面向对象方法和思想已被广泛接受,从 20 世纪 90 年代开始,人们开始将面向对象思想方法引入 RDB,对其进行基于对象的扩展,建立对象关系模型(Object Relation Data Model,ORDM),其着眼点是将复杂类型数据和继承机制引入关系数据管理过程当中。ORDB 作为 RDB 的对象扩展,本质还是基于“关系”的,这主要表现在基本数据单元还是“元组”(此时称为“行”或“结构”)和关系表,只不过通过引入复杂数据类型突破 1NF 限制而使得元组和关系可以相互嵌套,再通过引入“继承”和“引用”等增添了面向对象过程基本功能,作为常规 SQL 的超集,相应数据操纵语言是 SQL3 和 SQL2003。

4.2.1 RDB 基于对象扩充

按照面向对象思想，整数、浮点数、字符及布尔数等高级程序设计语言中的基本数据类型实际上也是面向对象技术中的“类”或“类型”，这是因为它们本身具有唯一标识（文字字面值）、操作运算（四则运算、字符串运算或布尔运算等）和操作与属性的封装（运算过程对于用户透明）。另外，按照数据模型的概念，这些基本数据类型还具有明确的数据关系（如数的“大小”关系等）、数据操作（如加减乘除和模运算等）和相应的数据约束（如两个整数做除法，分母不能为零，结果一定是整数等），因此，它们实际上也可看作数据库意义下的某种数据模型。只不过它们涉及的数据元素非常基本、数据关系比较简单和数据运算具有普遍意义而没有显式纳入面向对象方法中的“类（类型）”及数据库原理中的“数据模型”框架当中。但从中可以体会到，建立数据库意义下的“数据模型”和建立面向对象平台上的“类（类型）”，从系统实现的技术角度来看，都是构建相应的“数据类型”。另外，需要注意的是，在对象数据库中，通常可以将“（数据）类型”看作一种最基本的“类”。

对象关系数据模型（Object Relational Data Model，ORDM）作为关系数据模型扩充，着眼点是为了适应新的应用需求。面对各类新领域需求，首先就是扩充或构建新的复杂数据类型，以提升关系数据模型对于现实中较为复杂的实体，如多媒体和网络数据缺少实际模拟和抽象描述能力。人们正是从数据类型入手，基于面向对象方法，对传统的关系数据模型进行扩充，提出了 ORDM。这种扩充突出特点是突破 1NF 限制，并使用了面向对象的重要思想和基本技术。

（1）从内置数据类型上来看。通过引入新的数据类型对原有预置数据类型进行扩充，通过扩充的原子数据类型和相应构造器，实现了属性值的聚集（多集和数组）数据类型表示。

（2）从属性变量取值来看。通过引入结构（行或元组）数据类型将关系表中数据项（属性值）看作具有自身同型结构的数据实体，使得属性变量可以取值为另一个关系，实现了关系表的嵌套结构。

（3）从操作和属性的封装上来看。通过引入抽象数据类型实现了数据实体对象的封装，提供了防止引用数据和调用方法过程中子类对超类不合法改动的功能。

（4）从数据结构上来看。通过引入对象标识实现了具有强大语义表达能力的继承机制和由于嵌套结构导致数据结构的分层表示。

（5）从关系模型仍然是基本支撑角度来看。由于借鉴的面向对象方法更加接近于客观世界中的真实表示，因此，在 ORDM 中，扩充的 E-R 模型即 EE-R 模型中许多概念，如实体标识、多值属性和泛化/细化等，无须经过变换转化就可直接使用。

上述前三点使得对象关系数据模型具有了“对象”的基本特征，以下主要基于这三点讨论 ORDM。

4.2.2 对象关系数据类型

由 RDB 中数据类型扩展到 ORDB 中数据类型实际上包含两种情形：一种是当属性取值为原有数据类型时数据容量值的扩充；另一种是属性取值为 1NF 时限制的解除。

1. 大数据类型

一个系统需要有基本的预置数据类型。RDB 中预置数据类型不仅种类较少，同时各个

预置类型数据值本身容量较小，这从逻辑上来看也是合理的，因为属性值只是一个数据项，不需要显式地描述相关内容，其语义通常可以另行解释。如同一个计算机文件的标题那样，从存储代价上考虑不需要分配过多的存储空间。但 RDB 扩充到 ORDB 时，情况可能就会发生变化。ORDB 的主要应用对象是多媒体数据和网络数据（XML），从本质上它们都没有越出 RDB 原有预置数据类型的范围。例如，XML 数据可以看作是字符串类型数据，多媒体可以看作是二进制类型数据，将它们作为某种意义下的属性取值而存入数据库应该是相当自然的考虑。此时的问题只是将 XML 文档作为数据看待，相应字符串具有丰富的语义内容；而将多媒体对象作为数据看待时，相应二进制文件具有完整的声像含义。丰富语义和完整声像其数据容量也就不可能满足原有数据容量限制。另外，当属性取值为新引入的各类复杂类型数据后也有解除数据容量的需要。由此看来，ORDB 在保留 RDB 原有预置数据类型基础上，还需要引入具有较大数据容量的预置数据类型，这就是“大对象”类型数据。

大对象（Large Object，LOB）类型是 ORDB 中引入的具有较大存储容量的数据类型，其中的数据容量至少需要达到 2GB 以上，这主要是基于 XML 文档、多媒体数据及各类新的复杂数据取值的实际需求，其基本点是将 XML 文本或多媒体文件（图形、图像、音频、视频等）作为单个属性值进行存储管理。

ORDB 中的 LOB 类型一般分为如下两种子类型情形。

（1）字符型大对象数据类型（CLOB）：用于表示网页、XML 及其他一些字符型大文件。

（2）二进制大对象数据类型（BLOB）：用于表示视频、音频和图像图形等多媒体二进制数据文件。

2. 面向对象类型扩充

RDB 原有的数据类型难以表达各类新型数据应用的需求，而属性取值的 1NF 规则又大大限制了关系数据的语义表达能力。因此，引入复杂类型数据以增强系统内的语义表达能力实际上就是需要解除 RDB 关于属性取值的原子性 1NF。

注意到 RDM 之所以能够提供系统级别上统一的数据处理机制，其中一个内在缘由就是简单的数据类型和属性取值原子性要求。因此，引入复杂类型数据之后还需要引入针对其相应的函数操作功能，即需要将某些复杂类型数据与相应的处理函数绑定，从而自然地需要引入面向对象方法中的封装机制。

从技术实现角度而言，在 SQL3 或 SQL2003 中，解除 1NF 是引入聚集类型和行类型实现，而复杂类型数据与相关函数绑定则通过引入抽象数据类型实现。

此外，由于数据可以取值为多层嵌套与组合的更为精细庞杂的复杂类型数据，为了应对层次间各类数据对象唯一识别要求及提高系统处理效率，还需要引入“引用（指针）”类型数据。

由此可见，RDB 基于面向对象的数据类型也就是 ORDB 数据类型，主要可以划分为聚集类型、行类型、引用类型和抽象类型等几种情形，如图 4-2 所示。

需要说明的是，图 4-2 只是相对于 RDB 中新增添的数据类型，实际上 ORDB 应当兼容 RDB，因此 ORDB 中的数据类型集合是 RDB 中数据类型集合的超集，即包含了 RDB 中原有的各种基本数据类型。

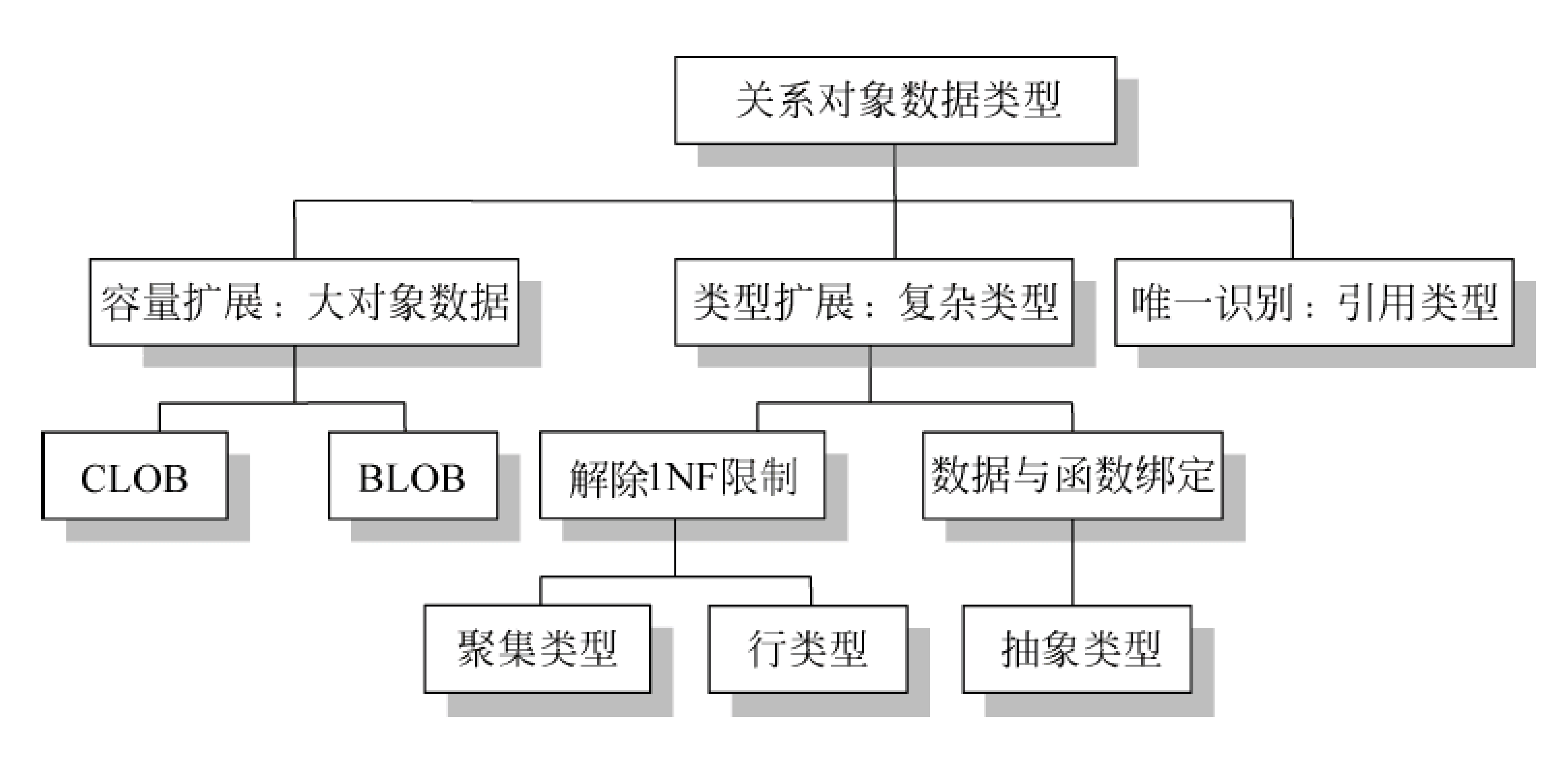


图 4-2 ORDB 中的数据类型扩充

3. 聚集类型与行类型

如前所述，1NF 对于 RDBMS 的研制和实现具有重大的意义，但这也是 RDB 难以适应新型数据库应用的要点所在。实际上，对于 RDB 的各类扩展通常都是由解除 1NF 为出发点的。

1) 解除 1NF 限制

RDM 基本要求是其中的关系模式满足第一范式即 1NF。1NF 对于银行、商店和酒店等的事务数据处理是必要的与合理的，但对于其他一些情形却力不从心。例如，当一个用户将数据实体看作一个对象的集合而不是元组集合时，由于对象中静态属性取值可能是多值的（如一个人的多个电子邮箱等）和引用的（如一个在职博士研究生同时也是计算机学院的讲师等），因此如果按照 1NF 要求，这些对象就可能需要由多个不同的元组或多个不同的关系表进行描述。由此会带来如下问题。

(1) 裂解数据语义。一个数据对象具有的完整语义信息被分解为多个满足 1NF 要求的关系元组，这并不符合人们的直观感受。

(2) 增加属性域管理开销。非 1NF 转换为 1NF 的基本途径就是增加属性列，由此将会增加更多的属性（数据）域，并带来更大的属性域完整性方面的开销。

(3) 加大数据冗余风险。通常属性取多值的关系表在 1NF 框架下需要满足 4NF 条件，此时有可能出现更多的在非 1NF 情形下不会出现的数据冗余问题。

(4) 需要更多的连接操作。4NF 要求必然导致更多更小的关系表存在，在实际数据操作时必须进行更为频繁的连接操作，而连接是关系表查询操作中的最大开销，对系统效率影响很大。

实际应用中，非 1NF 可以体现在以下两方面。

(1) 属性取多个类型相同的值。例如，一个人的通信联系为“办公电话、住宅电话、移动电话、传真”方式，这里，所有数据项可以在同一数据类型中取值。此时可以看作属性变量取值于聚集类型。

(2) 属性取多个类型可以不同的值。例如，一个人的住址数据为“街道、城市、邮编”方式，其中各个单元项在不同数据类型中取值。此时可以看作属性变量取值于结构类型。

解除 1NF 限制是对数据类型本身的实质性扩展，主要由引入新的数据类型实现。在 SQL3 和 SQL2003 中，通过引入用户自定义的“行（结构）类型”实现了属性的复合取值，通

过“聚集(集合体)类型”实现了属性的同型多项取值。

2) 聚集类型

聚集类型(collection type)也称为集合体类型,它是由一组类型相同的元素组成的满足一定要求的集合,具体可分为数组、列表、多集和集合 4 种情形。聚集类型可以看作 C/C++等中的数组类型在 ORDB 中的实现,其意义在于从逻辑上将具有相同数据类型的数据集合看作是具有“单个”数据域的数据实体,在物理上对如此数据集合进行整合存储管理。在 ORDB 数据操作语言中,聚集类型主要是数组和多集类型。

(1) 数组类型。数组类型(array type)是指相同类型元素的有序集合,数组大小需要预先设置。实际应用中,将具有相同类型数据值排序往往是必要的。例如,一本出版物如果有多名作者,则第一作者、第二作者等的区分就具有明显的意义。又如,一本著作的作者名可以以数组“John,Raul,Mary,White”形式表示,这使得人们可以区分各个作者在该书中的贡献和所担负的责任。

(2) 多集类型。多集(multiset)类型是指相同类型元素的无序集合,但允许一个元素出现多次,也称为包类型(bag type)。例如,一个小组五名同学的数据库课程考试成绩可以表示为一个多集类型数据{75,80,80,70,80}。这里,数据中元素相互间顺序无关紧要。如果要强调顺序,就转回到了数组。

需要注意的是,聚集类型并不复杂,当系统引入后述的结构(行)类型后,没有聚集类型也不会影响其实际使用。对于系统而言,将聚集类型作为单分量的结构类型或单列表进行处理即可,但这样做对于用户而言却不够直观自然。

3) 行(结构)类型

行类型(row data type)也称为结构类型(structural data type)、元组类型(tuple data type)或对象类型(object type)。其表现为一个元组(结构或行)且可以多次交替出现。行类型在逻辑上对应于“对象”,在关系表中表现为元组或记录。

行类型数据具有“数据项可不同型”和“元素间可嵌套”的特点。

(1) 数据项可不同型。行类型数据中的各个数据项可以具有不同的数据类型值,既可以是合法的各种数据类型值,还可以是另一个行类型值。例如,日期(1,April,2017)是由日、月、年份 3 个不同类型数据值构成的行类型数据。由于结构类型是传统关系模型中元组概念的对象推广,因此基本要素是其中的属性名和相应属性域。需要注意的是,行类型中给定元素属性域只能包含在某个确定的数据类型当中。

定义由成分属性 firstname 和 lastname 组成行类型属性 name。

```
Create row type Name as
 (firstname varchar(20),
  lastname varchar(20))
  final
```

定义行类型(属性)address。

```
Create row type Address as
 (street varchar(20),
  city varchar(20)
  zipcode int)
  not final
```

上述两个定义式中，final 和 not final 与子类型有关，在 SQL3 中，“final”表示行类型“Name”不能具有子类型，而“not final”表明行类型“Address”可以具有子类型。

(2) 元素间可嵌套。注意到元组的集合就是关系，因此结构类型中某个元素取值也可以是另外一个关系，此时整个关系称为嵌套关系(nested relation)。嵌套关系通常具有“关系—属性值—行(关系)—属性值—行(关系)……”的链接形式。嵌套关系模式如图 4-3 所示。

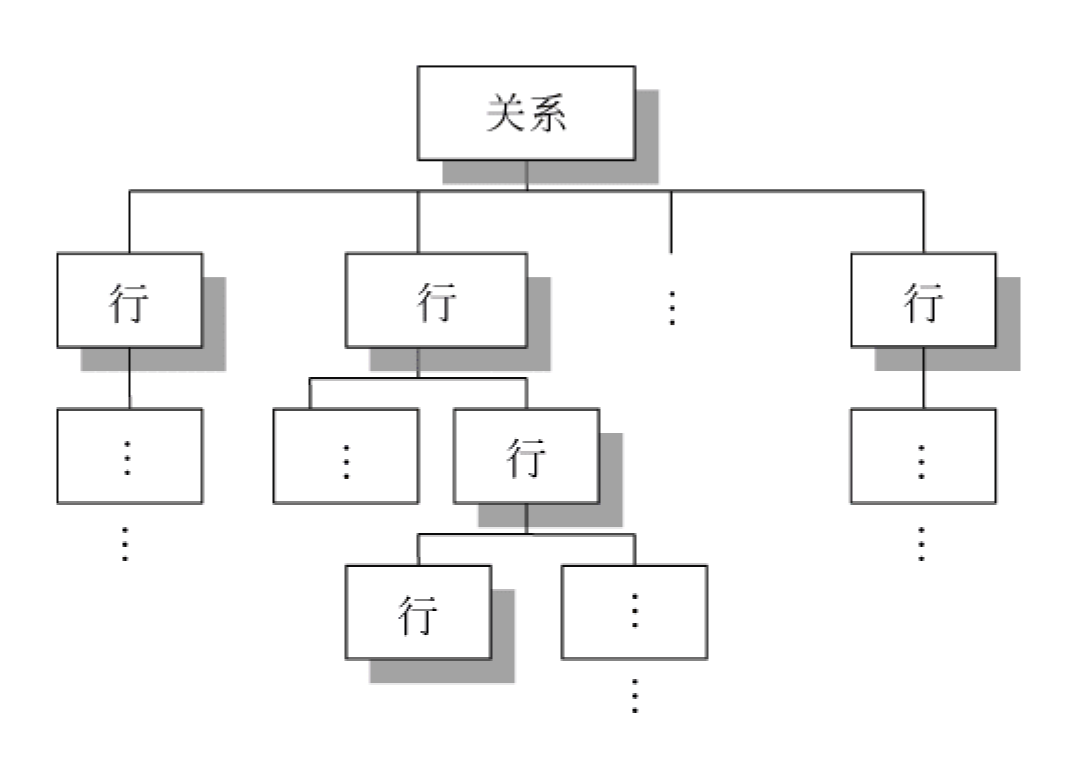


图 4-3 嵌套数据模型结构

作为 RDB 的对象扩充，如同关系表示给定域约束之后的同型元组集合，ORDB 中关系也就是给定域约束之后的行的集合。在数学的观点之下，单元素集合和组成该集合的元素具有一一对应关系，因此是等价的。由此出发，可以将单个的行类型数据也看作是一个关系，而将关系看作是行类型数据的聚集，在不至于混淆情况下，将 ORDB 中的关系表看作是关系类型数据，此时的“关系类型”的逻辑内涵实际上就是由其组成元素——行类型数据所界定。这样实际上是将数据库中不同粒度的数据元素都统一看作是同等逻辑层面上相应类型的数据，此等“有意混用”会在问题的分析与处理过程中带来相当的便捷。

【例 4-1】 在教育系统中，大学(University)与教师(Faculty)组成如下非平面关系。

```
University(uno, uname, ucity, staff(fno, fname, fage));
```

其中，属性 uno、uname、ucity 和 staff 分别表示学校编号、校名、所在城市和教师，而属性 fno、fname、fage 分别表示教师编号、教师姓名和年龄。这里属性 staff 取值为一个行类型数据集合，即另一个关系表，表示一所大学的所有教师，关系 University 就是一个嵌套关系。

类似程序语言，可用类型定义和变量说明方式描述嵌套关系结构，以下以伪码方式讨论常用的 3 种方式。

① 先定义关系类型，再定义关系数据。

```
type UniversityRel = relation(uno: string,
                              uname: string,
                              ucity: string,
                              staff: FacultyRel);
type FacultyRel = relation(fno: string,
                           fname: string,
                           age: integer);
Persistent var university: universityRel;
```

这里组合关系用持久变量(persistent variant)形式说明,供用户使用。

② 先定义行类型,再定义关系类型。

此时,定义元组类型 UniversityTup 与 FacultyTup,然后定义关系类型 UniversityRel 与 FacultyRel 分别为 UnivertyTup 与 FacultyTup 的集合,这种方法将更加灵活。

```
type UniversityTup = tuple(uno: string,
                           uname: string,
                           ucity: string,
                           staff: FacultyRel);
type FacultyTup = tuple(fno: string,
                        fname: string,
                        fage: integer);
type UniversityRel = set(UniversityTup);
type FacultyRel = set(FacultyTup);
```

③ 不通过定义关系类型,直接使用集合 set 形式。

```
type UniversityTup = tuple(uno: string,
                           uname: string,
                           city: string,
                           staff: FacultyTup);
type FacultyTup = tuple(fno: string,
                        fname: string,
                        fage: integer);
Persident var university: set(UniversityTup);
```

4. 抽象数据类型

对象的基本特征是状态(属性与联系)和方法(操作)的封装。但数据库并不完全等同于一般的软件工程设计,其中大部分数据元素并不需要自定义函数,以及这种函数与状态的封装,否则还会带来诸如安全性等方面的各种问题。在数据库中,对数据的查询和更新是系统提供的 public 操作,隐含在系统当中,不需要对每个对象或类各自定义一套操作函数。实际上,如果用户可以不受限制地自行定义数据操作,数据库安全性将得不到保障。但随着数据库应用范围扩展,常规数据库中的数据类型不能适应所有应用需求,常规数据操作难以满足新引进的数据类型(图形图像、音频影像和网络数据等)处理需求,需要有更为复杂的方法。对系统来说,不可能为这所有的特殊需求都一一定义相应的系统级的数据操作。此时,为满足一些新型复杂数据类型管理,就需要引入一种新的数据类型——抽象数据类型(Abstract Data Type,ADT)。

ADT 定义类似于面向对象方法中的“类”定义,ORDM 引入 ADT 的意义有以下两点。

① 可根据应用需要扩展特定数据类型并定义相应操作。

② 通过对象封装性,只有规定的函数可对其操作,防止各类错误操作对系统的破坏。

在理论上,ADT 需要和对象属性进行封装,但封装会引发一系列技术上的改造,可能会涉及 ORDB 的关系内核,同时由于实际应用中只需要对个别数据对象定义不同于系统统一提供操作的数据运算,因此 ORDB 中的 ADT 并不需要面向对象框架下的那种严格限定。实际上,此时的 ADT 不能算作面向对象思想中的“方法”,而只能看作“函数”,即动态操作和静态属性在逻辑上是“分离”的。例如,ORDB 中的结构类型可以看作“对象”,而作为函

数的 ADT 通常都和结构类型进行了必要的区分。

① 结构类型不定义相关操作：因此，结构类型是 ORDM 中更为广泛使用的类型。

② ADT 创建中不定义结构类型：ADT 只定义属性(分量)类型，在应用中受到必要限制，只是作为结构类型的一种补充。

如上所述，在 ORDB 中 ADT 只是作为函数出现，定义 ADT 就是定义相应的各类函数，这主要分为“内置函数”和“用户定义函数”两种情形。

1) 内置函数

ADT 属性部分定义与结构类型相同，但所有 ADT 都应具有系统提供的、适用的公共函数。在 SQL3 中，ADT 可以调用的公用函数有下述 3 种内置函数。

(1) 构造器函数(constructor function)。用以生成一个属性值为给定值的 ADT 对象，其语法格式为：<类型名>([<需要生成对象的属性值>])。

如果没有给出对象属性值，则构造函数生成一个属性没有被初始化的对象< ADT 名>()，但具有相应对象标识。

(2) 观察器函数(observer function)。对于类型 T 的每个属性 A，观察器函数 T. A()读取 T 关于 A 上的值，即 T. A()返回 T 的属性 A 的值。

(3) 变更器函数(mutation function)。用于删除或修改对象属性，其格式与结构类型相同，功能是将某属性值设置成一个新值。该函数具有潜在破坏性，在访问控制中需要对其授权进行必要的限制。

SQL3 允许这些内置函数锁定公共应用，并使用 EXECUTE 特权访问。

2) 用户定义函数

除了所有 ADT 都可以调用的内置公用函数外，ADT 的自定义函数分为内函数与外函数。

(1) 内函数(internal function)。定义在类型创建语句 CREATE TYPE 内的函数，其语法格式为：

```
FUNCTION <名称>(<参数及其类型>)
RETURN <返回类型>;
<变量说明>;
BEGIN
<函数体>
END;
```

内函数通常使用扩充了的 SQL 语句编写，适合于定义较小的操作方法。

(2) 外函数(external function)。在类型创建语句中定义调用声明，函数实现则置于类型创建语句之外。外函数除了可用 SQL 语句编写外，还可用其他程序设计语言编程。外函数具有较为严格的限制，其调用函数必须为内函数或系统内置函数，不能调用非 DBMS 定义的函数，以便防止非法函数对系统的干扰和破坏。外函数声明格式为：

```
DECLARE EXTERNAL FUNCTION < signature >
LANGUAGE < language name >
```

由此可知，在外函数声明中，除了调用声明外还需要说明其使用的编程语言。

5. 引用数据类型

行类型会带来数据对象与属性值之间的相互嵌套。从面向对象方法角度考虑，这种属性值对相应对象(行类型)的调用应当通过 OID“间接”进行。继承机制中子类对于超类属性值调用也是如此，需要通过相应 OID 进行。实际上，如果直接调用数据实体，就有可能带来语义上的混乱，造成无穷嵌套。例如下述两个关系：

```
UniversityRel(uno, uname,ucity, staff(FacultyRel));
FacultyRel(fno,fname,fage,works - for(UniversityTup))
```

上述定义中分别涉及行类型 UnivesityTup 和 FacultyRel(本身是一个关系)，即在关系 universityRel 的元组(UnivesityTup)中包含关系 FacultyRel 的成分；而关系 FacultyRel 的行类型(FacultyTup)中又包含行类型为 UnivesityTup 的成分。此时类型构造示意如图 4-4 所示。由于会引发无穷嵌套，这种“直接”调用数据实体的情形通常不能使用。

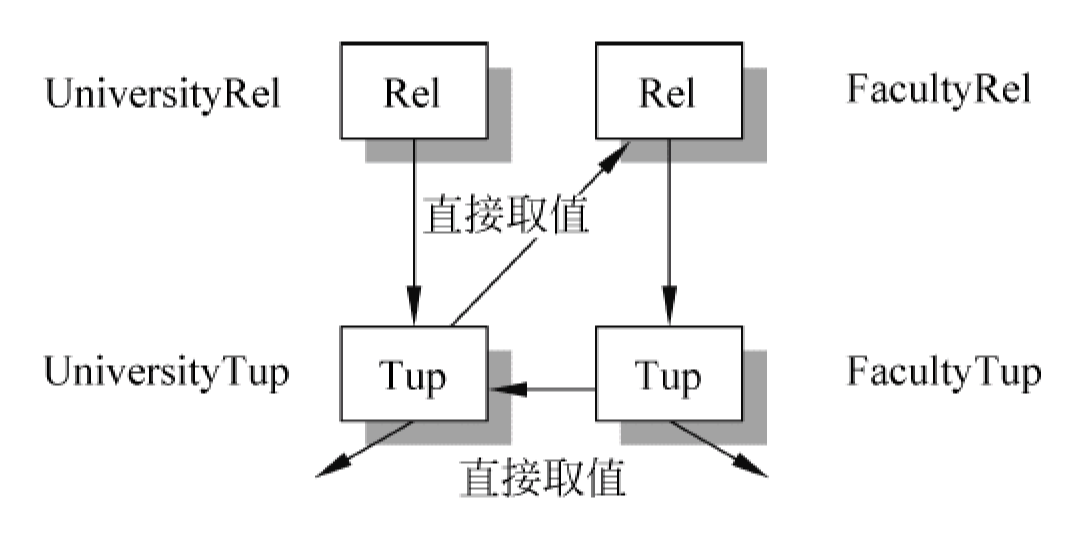


图 4-4　递归引发无穷嵌套

引用(reference)是一种“间接”调用数据实体的技术，即当属性变量取值为一个结构类型时，系统存储和管理相应类型中的数据对象的对象标识即 OID，将类型中的实例映射到类型中的 OID，通过 OID“间接”提供有关对象的细节抽象。使用引用机制避免数据调用过程中的“无穷嵌套”问题。

前述行类型 UniversityTup 中有一个属性 staff 的数据类型是关系类型 FacultyRel，在实现时可以不采用原有实例调用方式，而是采用“引用方式”(指针方式)，用指针指向关系类型 FacultyRel 中各个有联系的职工。元组 FacultyTup 中有一个属性是行类型 UniversityTup，实现时也采用“引用方式”。图 4-5 所示的是采用“引用”类型后的类型构造示意图。图中用虚线表示“引用”类型，实线表示类型与成分相连。

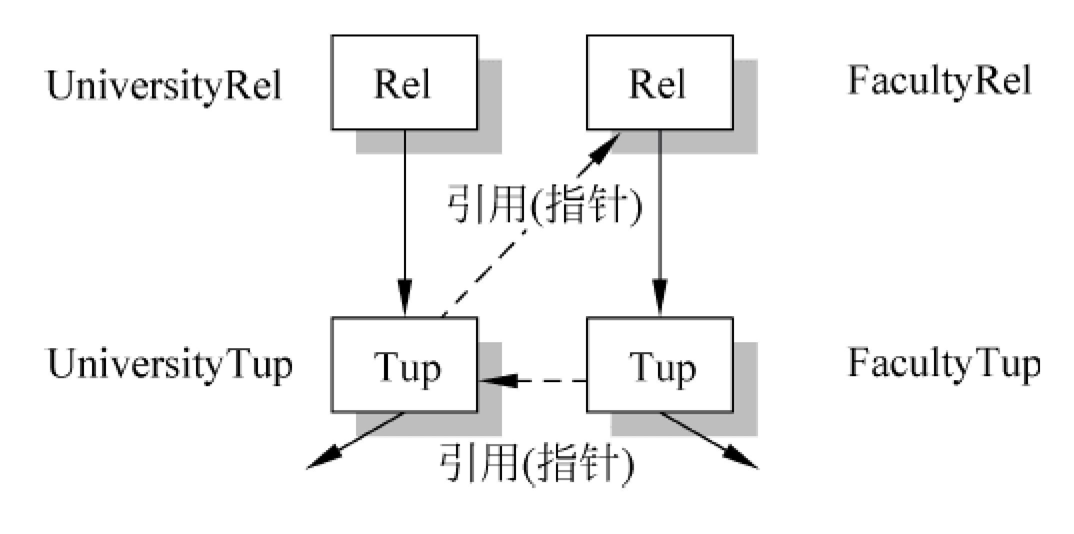


图 4-5　采用“引用”概念的类型构造

由此可知，如果类型中一个对象的属性取值为引用，则该属性值就是被引用对象的 OID 而非该对象的实际取值。在面向对象方法中，OID 对于用户来说透明，是一种系统码，但在 ORDB 系统中，允许用户以引用类型 REF 访问 OID，但仅限于访问而不可修改。

4.2.3 继承机制

按照面向对象方法，行对应于对象，而同型对象构成所在类。RODM 中最基本的数据结构就是类之间的相互关系。在面向对象观点之下，这种类的相互关系的基本语义特征就是“一般/具体”，即“超类/子类”关联。因此建立了行类型之后，还需要讨论相应的类继承在 RODB 中的技术实现机制。

“一般/具体”是讨论不同类型之间关联的出发点，这在面向对象方法当中也称为“泛化和细化”。不同类之间的“超类/子类”关联在理论上不仅体现了泛化/细化理念，对涉及演化问题的数据系统，还在技术上避免重复定义，减少数据冗余，实现重用与联编，提高系统效率。

1）泛化和细化

泛化和细化(generalization and specialization)是对概念间联系进行抽象的一种方法。当较低层面上的抽象表达了与之联系的较高层面上的抽象的特殊情况时，则称较高层面上抽象是较低层面上抽象的“泛化”，而较低层面上抽象是较高层面上抽象的“细化”。这种细化联系是一种“是”(is a)的联系。在具有泛化和细化的对象类型之间，较高层面上的对象类型称为“超类型”(supertype)或“基类型”(basetype)，较低层面上的对象类型称为“子类型”(subtype)或“派生类型”(devoidedtype)。

2）继承与继承分类

在继承机制中，子类型继承其超类型的属性和函数，同时子类型本身还具有其他的特定属性和函数。

(1) 按照子类是否继承超类的所有属性和函数，可将继承分为全继承和部分继承。

(2) 按照一个子类是否具有多个超类，可将继承分为多继承和单继承。

(3) 按照子类的父类是否还有父类，可以将继承分为链继承和非链继承。

部分继承、多继承和链继承可表示更为丰富的语义内容，更加适合于实际中的应用情形，但技术处理比较复杂。

在 ORDB 中所需要讨论的继承机制课题主要有下述几点。

(1) 在部分继承中如何统一制定继承的语法格式描述。

(2) 在多继承和继承链中如何处理多个父类都具有同名属性和同名函数问题，这涉及面向对象方法中的精华——多态和联编等。

(3) 给出一个查询语句，在继承链中，是只查找子类中满足条件的对象数据，还是在涉及的所有类(子类或超类)中查找满足条件的数据。

4.3 对象关系数据模型

通过复杂数据类型(聚集与结构类型)，解除了关系数据模型的 1NF 束缚；通过抽象数据类型 ADT，部分实现了数据对象的自定义操作；通过引入继承机制，具有了派生和重用的功能。以这些基于对象的拓展为基础，就可将关系数据模型(RDM)拓展到对象关系数据模型(ORDM)。

4.3.1 PRDM 与 ORDM

从逻辑和技术角度来看，RDM 到 ORDM 的拓展可以分为两个步骤，先是将数据类型拓展的复杂数据类型得到“后关系模型”，再将后关系模型拓展为对象关系数据模型。

1. 后关系数据模型

后关系数据模型(Postrelational Data Model，PRDM)可通过将关系数据模型进行下述两方面扩充而得到。

(1) 元组属性取值可为聚集数据类型(数组、列表、多集和集合)。

(2) 元组属性取值可为结构数据类型，并且还可进行相应的嵌套取值。

需要说明的是，由于单个元组(行)和由其组成的单元素集合存在一一对应关系，因此，在逻辑上可以对“单个元组”和“元组组成集合”不加区分而“有意”混用，在此意义下，属性取值为“单个”元组就认为是取值于由该元组构成的单元素元组集合，因此就自然得到元组属性可以取值于“元组集合”即关系表。如果将同型结构组成的元组集合看作“关系表”或“关系类型”，此时，结构类型的嵌套可以统一描述为“属性可以取值于另一个关系类型”。

简而言之，引入了复杂数据类型(聚集和结构类型)的 RDM 就是 PRDM。

PRDM 结构如图 4-6 所示。

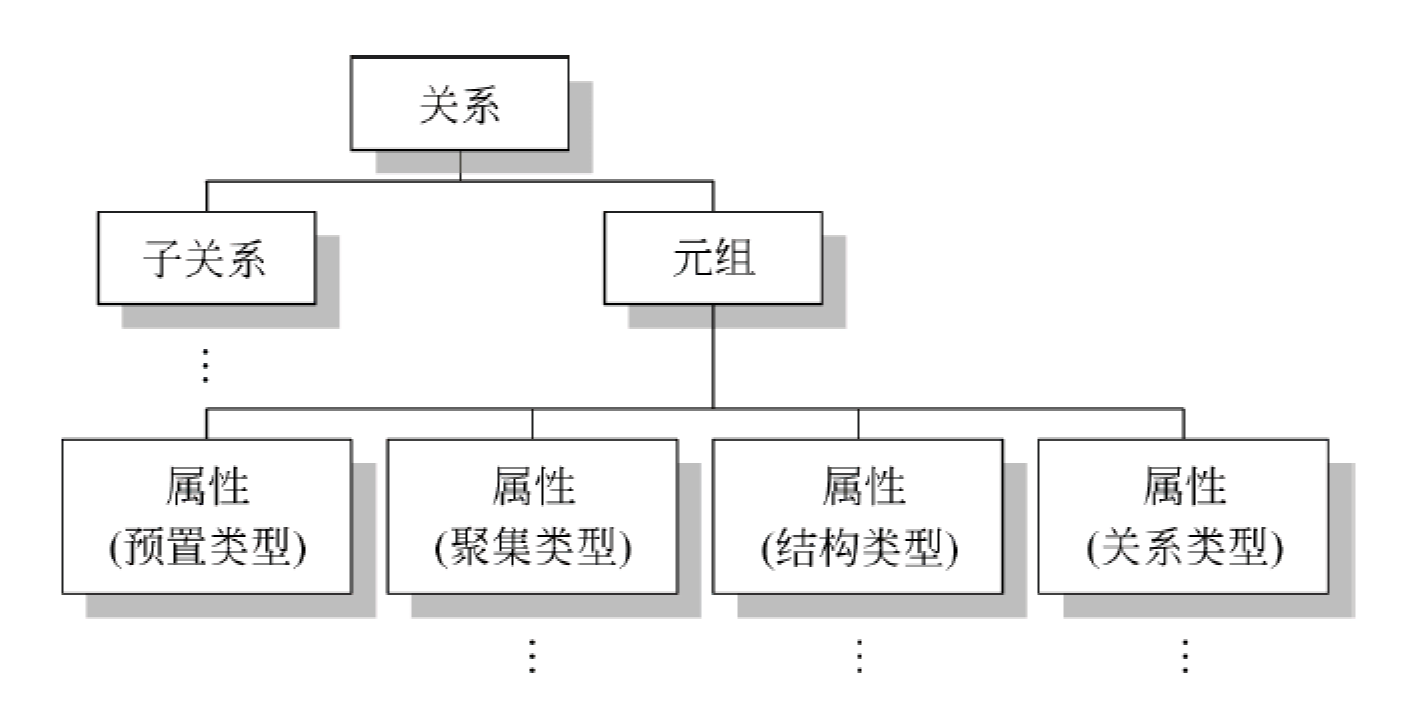


图 4-6 后关系数据模型结构

2. 对象关系数据模型

对象关系数据模型(Object-Relational Data Model，ORDM)可通过将 PRDM 进行下述两个方面扩充而得到。

(1) 引入“继承”机制以实现结构类型之间的“层次”关联。

(2) 引入元组对象标识和“引用技术”以实现递归过程中的“指针”取值。

如前所述，RDB 具有支持数据类型有限和不支持高级语言程序直接访问数据库两个明显弱势。ORDM 较好地解决了上述第一个问题，部分解决了第二个问题。ORDM 提供了更为丰富的类型系统(基本类型、聚集类型和结构类型)和面向对象的基本机制(对象标识、数据引用和类型继承)，在对关系模型扩充建模能力的同时保持了关系模型最重要的基础——关系表格结构和关系的非过程性访问。基于 ORDM 的 ORDBS 为需要使用面向对象特征的 RDB 用户提供了一个简单快捷的移植途径。

4.3.2 对象联系图

在关系数据框架内,可以有效使用实体联系图即 E-R 图;在基于对象扩展的对象关系框架内,也需要将 E-R 图扩充为对象联系图即 OR 图。

OR 图基本成分可以描述如下。

① 椭圆:表示对象类型(相当于实体类型)。

② 小圆:表示属性是基本数据类型(整型、实型、字符串型等)。

③ 椭圆之间的边:表示对象之间的嵌套或引用。

④ 单箭头(→):表示属性值是单值(可以为基本数据类型,也可以是另一个对象数据类型,即结构数据类型)。

⑤ 双箭头(→→):表示属性值是多值(属性值是基本数据类型,或者是另一个对象数据类型,即关系数据类型)。

⑥ 双线箭头(⇨):表示对象类型之间的超类与子类类型(由子类指向超类)。

⑦ 双向箭头(←→):表示两个属性之间的值的互逆联系。

图 4-7 所示的是一个数据库模式的对象联系图,其中有大学、教师、上课教材等信息。

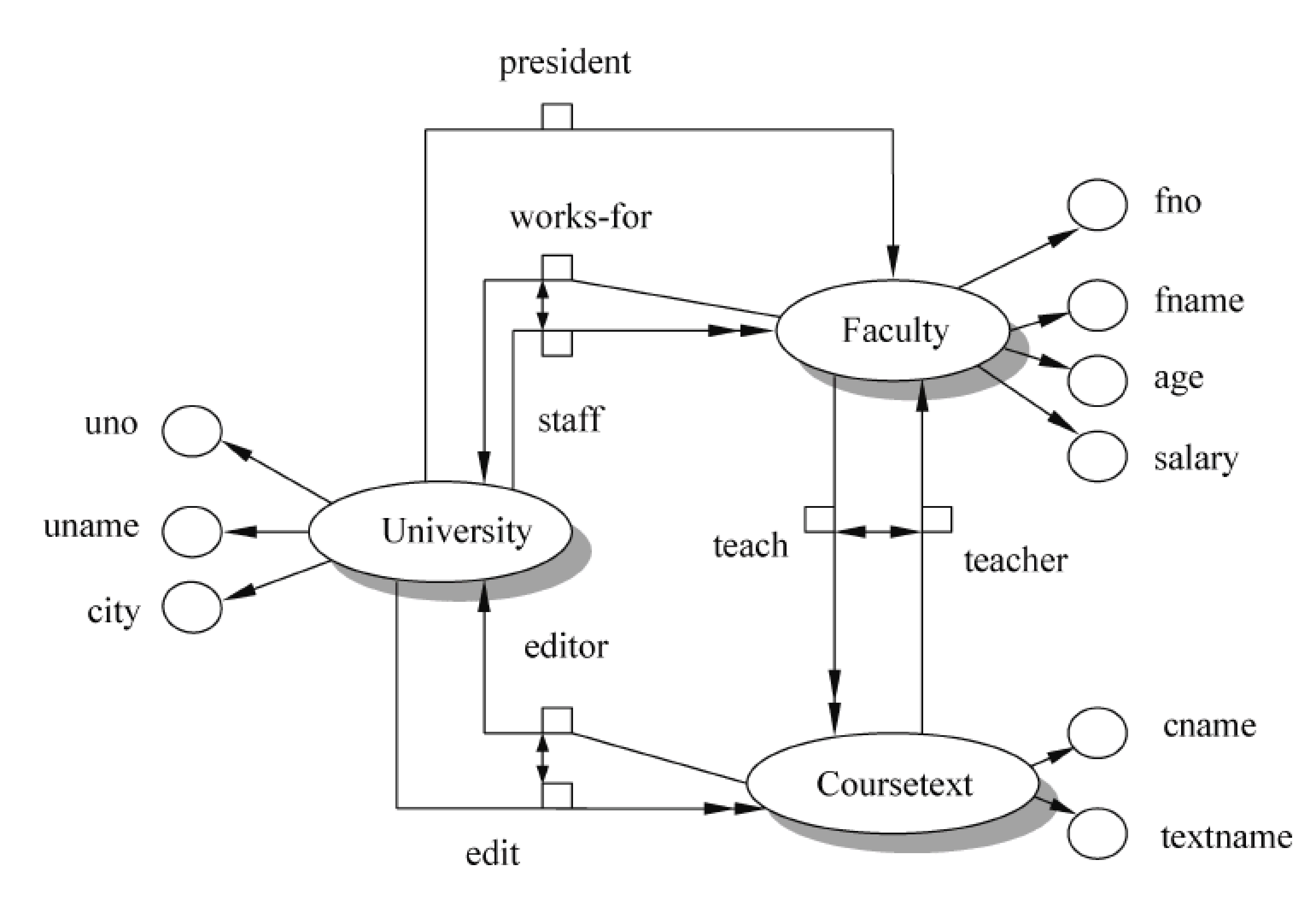


图 4-7 对象联系图

University:有关大学信息的对象类型,共有 6 个数据类型,其中学校编号(uno)、校名(uname)和学校所在城市(city)等是基本数据类型;属性校长(president)、staff 和 edit 是复合数据类型,其中 president 是单值属性,表示学校中有一位教师是校长, staff 和 edit 是多值属性,分别表示学校有若干教师,属性 edit 表示学校编写了若干本教材。

Faculty:有关教师信息的对象类型,共有 6 个属性。其中教师工号(fno)、姓名(fname)、年龄(age)和工资(salary)是基本数据类型;works-for 表示教师服务的学校,为单值属性,是复合数据类型;teach 也是复合数据类型,为多值属性,表示教师开了若干门课程。

Coursetext:有关课程与教材信息的对象类型,共有 4 个属性。其中两个是基本数据类型,即课程名 cname 和教材名 textname;还有两个数据是复杂数据类型,属性 teacher 表示

开课的教师，editor 表示教材编写的学校。

类型定义中的成分现在用从类型定义到值域类型的属性表示，如 teach 是一个从对象类型 Faculty 到对象集合(其成分是 Coursetext 类型)的属性。属性之间的双向箭头(←→)表示两个属性之间的联系为互逆联系。例如，teach 和 teacher 是一对互逆的属性，此处，teach 是多值属性，teacher 是单值属性，实际上体现了 Faculty 与 Coursetext 之间对象的 1∶n 关系。

【例 4-2】 图 4-8 所示的是一个带泛化/细化联系的 OR 图。对象类型 Person 是一个超类型，有属性 name(姓名)和 age(年龄)。对象类型 Faculty 是 Person 的一个子类型，自动具有 name 和 age 两个属性，表示"每个教师是一个人"的语义。但子类型 Faculty 还可以比超类型 Person 有更多的属性，如 fno(工号)、salary(工资)等。

对象类型 Student 也是 Person 的一个子类型。自动具有 name 和 age 两个属性，同时，自己还有属性 sno(学号)。在图 4-8 中，泛化/细化联系用泛化边(双线箭头)表示，泛化边从子类型指向超类型。

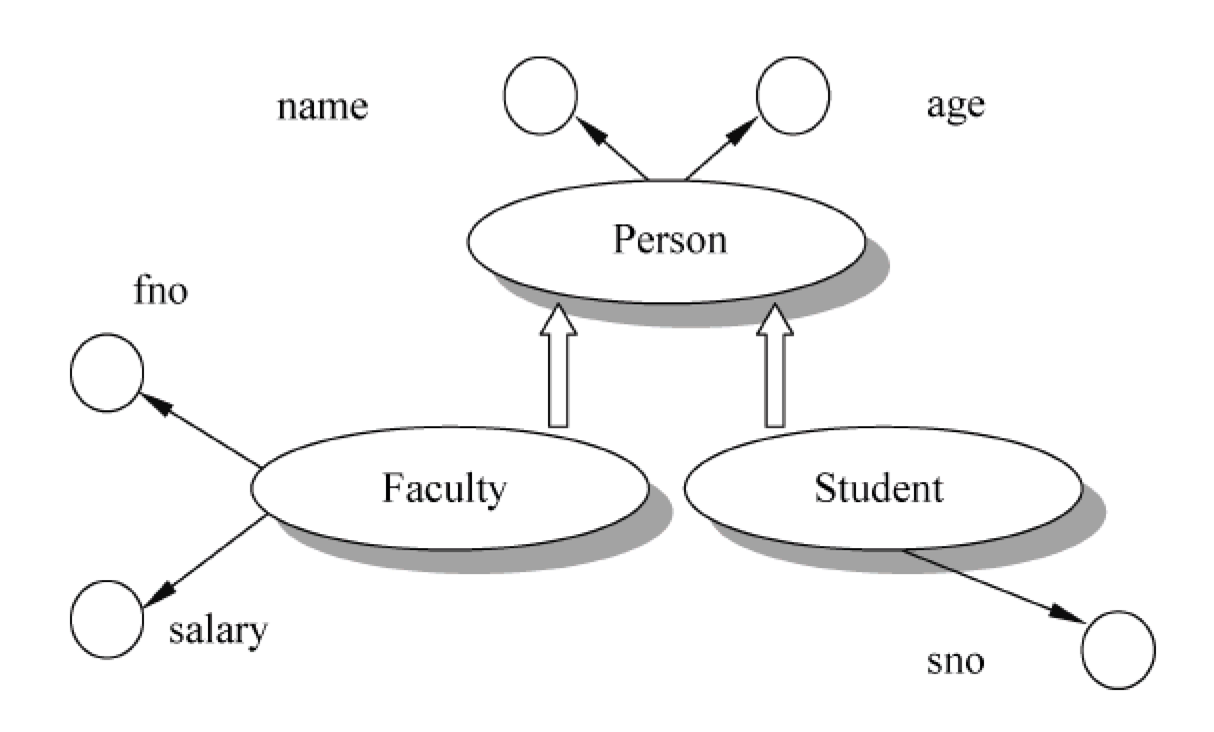


图 4-8　带泛化边的对象联系图

OR 图是描述对象关系数据模型的基本工具，它不仅完整揭示了数据之间的联系，也把查询的层次观点表现得相当清楚。例如，一个查询可能是从 University 开始，把 Faculty 看成是它的子对象(通过值为集合的函数 staff)。这样，查询形式如同数据库中有嵌套关系 University(…,staff(…))，这里的子关系 staff 包含了一所大学的所有教师信息。但是，另一个查询可能正好相反，那么就要用相反的层次观点解释。例如，从 Faculty 开始查询，把 University 作为子对象，通过单值函数 work-for 来实现，如果数据库中有嵌套关系 Faculty(…,works for(uname,city,…))。任何形式的层次联系均被包含在对象联系图中，而且实现时不会有冗余现象。

研究 ORDM 后就需讨论相应的数据库系统操作语言，如同 ORDM 是 RDM 的扩充，相应的 ORDB 语言也应当是 SQL 的扩充。

4.3.3　对象关系数据库语言 SQL3

在 20 世纪 80 年代中期，随着面向对象技术的兴起，人们开始在传统 SQL 语言基础上加入面向对象内容。

早期对象关系系统 POSTGRES 在 1986 年由美国加州大学伯克利分校开发，Illustra 是 POSTGRES 的商业化版本。

惠普公司在 1990 年推出的 Iris 系统，支持一种称为 ObjectSQL(OSQL)的语言。

1992 年，Kifer 等提出的 XSQL 也是 SQL 的面向对象扩充。在 SQL 标准的 1999 年版本中，增加了对象关系扩展，而在 SQL 的 2003 年版本中，也引入了具有对象关系特征的标准。

以下沿用通常说法，将包含对象关系功能的 SQL 称为 SQL3。

SQL3 标准包括下述 4 个主要部分。

1) 框架、基础、绑定和对象

这些基础部分主要是 SQL/框架、SQL/基础、SQL/绑定和 SQL/对象。

(1) SQL/基础处理新的数据类型、新的谓词、关系操作、游标、规则和触发器、用户自定义类型、事务管理和存储例程。

(2) SQL/绑定包括嵌入式 SQL 和类似于 SQL2 中的直接调用。

(3) SQL/对象包括新的数据类型，如二进制大数据对象(LOB)及大对象定位器、用户自定义数据类型、类型构造器、聚集类型、用户自定义函数和过程和触发器。SQL3 中的对象主要有以下两类。

① 结构类型或行类型，其实例就是“关系表”中的元组。

② 抽象数据类型(ADT)，其特征是通用类型，与“类”概念类似，主要用于元组分量。

2) 新部分寻址时序，SQL 事务管理

SQL 时序用于处理历史数据、时间序列数据和其他时序扩展，其基本标准是由 TSQL2 委员会提供。SQL 事务管理说明规范化供 SQL 实现者使用的 XA 接口。

3) SQL/CLI，即调用层接口

QL/CLI 提供某些规则，这些规则允许执行没有给出源代码的应用代码，并且避免了进行预处理的需要。它还提供了一种新的类似于动态 SQL 的语言绑定类型。

4) SQL/PSM，即永久存储模块

SQL/PSM 指定在客户机和服务器之间划分应用的设施，其目的是通过最小化网络流量来增强性能。

由此可见，增加了对象关系功能的 SQL 标准具有下述基本特征。

(1) 具有传统 RDB 标准 SQL 的基本功能。

(2) 具有定义数据类型与抽象数据类型的能力。

(3) 具有数据之间继承与组合的功能。

(4) 具有自定义函数和使用的功能。

正是由于这种 SQL 具有经典 SQL 明显特征，又进行了必要的面向对象扩充，适应于 ORDB 模式定义、数据操作和数据控制，因此称为 ORDB 的基本语言。现有 ORDB 基本都遵循相应 SQL 的工业标准。

如前所述，这就是 SQL3。本节主要讨论 ORDB 语言 SQL3 的一些最基本内容，如数据创建、数据查询与更新等。

4.4 对象关系数据创建

从技术实现角度来看，数据类型就是给定变量的取值域和相应操作运算，以便系统对于变量取值进行存储操作和有效处理。在常规关系中，基本操作单元或技术变量是元组，在对

象关系当中，通过引入嵌套机制等将“元组”扩展为“行类型”，此时的数据操作单元就是“类型”，因此，SQL3 的类型创建是 ORDB 语言的重要特色。

由系统原有的和新建的内置数据类型(扩展类型)就可创建用户定义类型，主要包括聚集类型、行类型及 ADT 和引用类型。SQL3 的数据类型如图 4-9 所示。

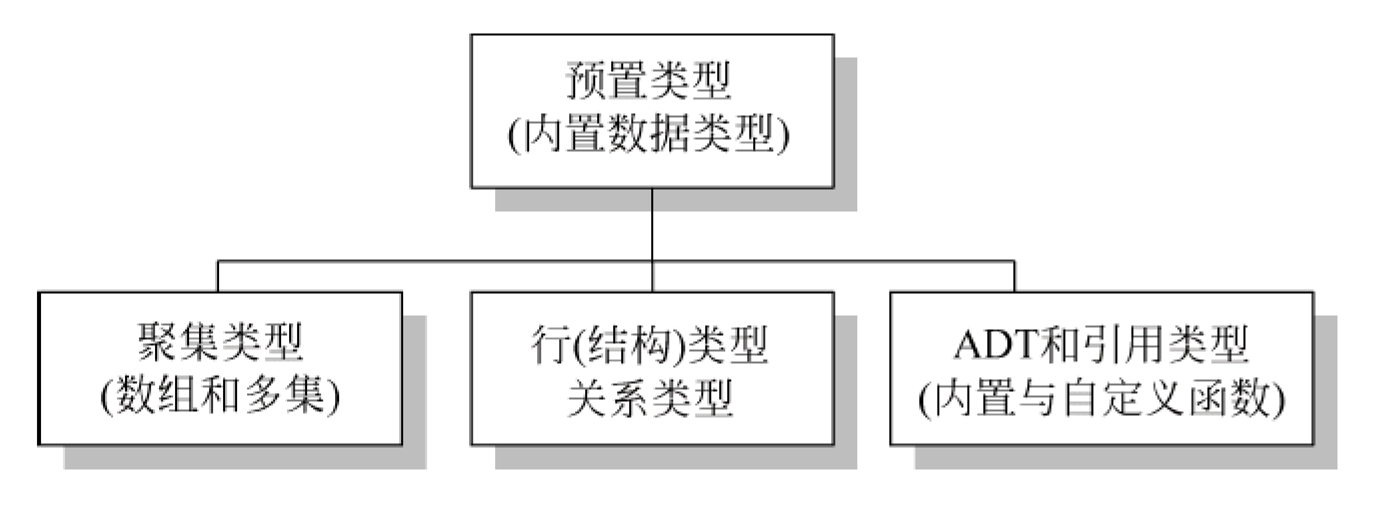


图 4-9　SQL3 的数据类型

4.4.1　类型创建

ORDB 数据类型较之 RDB 更为丰富，也比较复杂。首先需要明确新增加的预置数据类型，然后再基于内置数据类型创建复杂数据类型，即结构与聚集类型。另外，还需要创建由于嵌套行类型而需要的引用类型，同时还有面向对象方法中的继承机制。

1. SQL3 预置数据类型

SQL3 除了保留常规 SQL 原有的基本数据类型外，同时也在此基础上增加了 BOOLEAN、CLOB 和 BLOB 3 种新的基本数据类型。通过这些内置基本数据类型，就可以创建各类用户自定义数据类型。

(1) BOOLEAN 数据类型：这是一个真值类型，具有值域{True，False，unknown}，支持 not、and 和 or 3 种逻辑操作。同时，SQL3 还增加了两个新的操作：every 和 any。这两个操作的参数都是 BOOLEAN 数据类型，通常可由一个表达式得到。

(2) CLOB(Character Large Object)数据类型：这是一种长度不受限制的变长字符串，通常处理定位操作，主要用于存储长字符串数据。其功能类似于游标操作，可以将一般字符操作难以处理的字符串分别处理，同时支持“相等”与“通配 LIKE”操作。

(3) BLOB(Binary Large Object)数据类型：这是一种二进制串，通常处理初等算术操作，主要用来存储音频和图像数据。

【例 4-3】　BLOB 数据类型。

```
CREATE TABLE mail (origin VARCHAR(20),
                   address VARCHAR(20),
                   arrival DATE,
                   message BLOB(10M)
                   );
```

说明：属性 message 取值于二进制大数据对象类型 BLOB。大数据对象通常用于外部应用，通过 SQL 对其进行全体搜索是无意义的。在应用程序中，一般只查询大对象的“定位器”，然后通过定位器从宿主语言中操作该数据对象。

2. 行(结构)类型

SQL3 扩展类型系统中新增的行类型是其特色。行类型语法格式如下。

```
CREATE ROW TYPE <ROW name> AS(<component declarations>).
```

说明：此时的关键词 ROW 可以省略。

【例 4-4】 创建行类型。

```
CREATE TYPE Emp AS(name VARCHAR(35),
                  age INTEGER
                 );

    CREATE TYPE Comp AS(compname VARCHAR(25),
                      location VARCHAR(20)
                     );
```

创建相应的关系表：

```
CREATE TABLE Employee OF Emp
  VALUE FOR emp_id ARE SYSTEM GENERATION;

CREATE TABLE company OF Comp;
```

说明：行类型的对象标识由系统生成，上述子句“VALUE FOR emp_id ARE SYSTEM GENERATION”就表明了这一点。

【例 4-5】 生成行类型。

```
CREATE TABLE cust (cust# CHAR(4),
                   addr ROW (street CHAR(50),
                            city CHAR(25),
                            state CHAR(2),
                            zip CHAR(10))
                   PRIMARY KEY (cust #)
                  );
```

说明：上式中的 ROW 称为无名称行(类型)，因为是在关系表中直接定义，故不需要赋予其名称，因而得名。

3. ADT 创建

结构类型提供了将数据项属性值看作“对象”的功能，并允许通过行的嵌套实现复杂类型构造，但并不提供对象的封装机制。用户对于相应数据的特定操作通过 ADT 函数实现。

1) ADT 声明

抽象数据类型(Abstract Data Type, ADT)也称为用户定义数据类型，它是 SQL3 提供的类似于“类”的对象类型构造，通过 ADT，用户可根据需要自行定义带有自身行为说明和内部构造的用户数据类型(数组和多集)。ADT 定义包括下述基本内容。

① 关键字 CREATE TYPE。

② ADT 名称。

③ 关键字 AS。

④ 用“()”括起来并由逗号分隔的属性及类型列表。

⑤ 用逗号分隔的方法列表，此时的方法包括参数类型和返回类型。

由此得到创建 ADT 的一般格式为:

```
CREATE TYPE <ADT name> AS (属性名称及其类型,
                          EQUAL、LESS THAN 函数声明,
                          其他函数或方法声明
                          );
```

【例 4-6】 创建抽象数据类型 AddressType 和 StudentType。

```
CREATE TYPE AddressType AS(street CHAR(50),
                          city CHAR(20),
                          FUNCTION houseNumber()RETURN CHAR(10));

CREATE TYPE StudentType AS (name CHAR(30),
                            address AddressType
                           );
```

说明:在 SQL3 中,关键字 FUNCTION 需要后面紧跟方法名称和一个用括号把参数和参数类型都括起来的列表。本例抽象数据类型 AddressType 中,方法没有参数,但括号仍然必需。参数的表达形式为参数名称和参数类型,如(a INTEGER, b CHAR(5))等。在抽象数据类型 StudentType 中,元组的第二个元素本身还是一个抽象数据类型 AddressType。

2) 用户自定义函数

对于 ADT,用户自定义函数语法格式为:

```
CREATE FUNCTION <func_name> (<argument_list>)
   RETURN <type_name> AS (<file_name or SQL_repression>);
```

用户定义函数后就可在相应抽象数据类型创建中进行调用。在上述语法格式中,AS 之后是函数体,可以分为下述两种情形。

(1) 如果函数体是由 SQL 计算完备的扩展版本编写,可直接在 AS 后编写相应的 SQL 语句。例如,计算某个教师与另一个教师 Wang 的工资差函数定义为:

```
CREATE FUNCTION salary_differ (float)
   RETURN float
   AS SELECT $ sal_salary
     FROM teacher
     WHERE name = 'Wang';
```

(2) 如果函数体是由某种高级程序语言编写的,在 AS 后出现可执行代码的文件名称。

基于对象封装的考虑,ADT 中的属性和函数可以分为下述 3 类。

(1) PUBLIC:在 ADT 接口上可见。

(2) PRIVATE:在 ADT 接口上不可见。

(3) PROTECTED:仅在相应子类上可见。

在 SQL3 标准中,子类型继承父类型过程中可在子类中加入新的属性和操作,子类还可重新定义在其超类中已经定义的任何函数,但需要有相同的签名,这也就是对象方法中的函数“重载”。ADT 对于重载的支持主要体现在当其调用相同名称函数时,会对函数参数的个数和类型进行必要检查以确定最终需要调用哪个函数。对于函数而言,此时需

要注意以下问题。

① 如果一个函数被调用，最好的匹配是基于所有变元类型的选择。

② 对于动态连接，需要考虑运行时间的类型参数。

【例 4-7】 创建一个用户自定义函数。

```
CREATE TYPE person_type
   (PUBLIC
      name CHAR(20),
      address CHAR(60),
      sex CHAR(1),
      birthdate DATE DEFAULT now(),
   PUBLIC
      FUNCTION age (birthdate, 'now'),
      RETUENS INTEGER,
   )
CREATE FUNCTION age(birthdate,'now') for person_type
   BEGIN
   <the codes of this funciton>
   END
CREATE TYPE emp_type UNDER person_type
   AS (emp_id INTEGER,
      salary REAL,
      FUNCTION Give_raise (abs_or_pct BOOLEAN,amount REAL)
      RETUENS REAL
     )
   INSTANTIABLE
   NOT FINAL
CREATE FUNCTION Give_raise For emp_type
   BEGIN
      <the codes of this function>
   END
```

说明：抽象数据类型 emp_type 是另一个抽象数据类型 person_type 的子类型。除了继承 person_type 的属性和操作之外，emp_type 还有自身特有的属性；INSTANTIABLE 声明这个类型是实例化了，NOT FINAL 声明类型可以有子类型；最后还定义了该类型的方法。

4. 聚集类型

SQL3 通过 ADT 生成聚集(数组和多集)数据类型。

【例 4-8】 创建一个记录有关图书的数据类型。

```
CREATE TYPE Pblisher AS
  (name VARCHAR(20),
   branch VARCHAR(20))
CREATE TYPE Book AS
  (title VARCHAR(20),
   author_array VARCHAR(20)ARRAY[10],
   pub_date DATE,
   pulisher Publisher
   keywords_set VARCHAR(20)MUTILSET)
CREATE TABLE books OF Book;
```

说明：首先创建类型 Pulisher，包括两个属性 name 和 branch；其次创建结构类型 Book，其中，author_array 是数组类型（存储容量为 10），keywords_set 是多集类型；最后，定义由结构类型 Book 组成个关系表 Books。

【例 4-9】 创建上述数组 author_array 数据值集：

```
ARRAY['John', 'White','Black']
```

创建上述多集 keywords_set 数据值集：

```
MULTISET['computer','db','ordb']
```

创建行类型 Book 的元组如下。

```
('compilers', array['John', 'White','Black'],NEW Publisher('Springer','Berlin'), MULTISET
['computer','db','ordb'])
```

说明：上述元组创建中，通过适当参数调用 Pulisher 的构造函数 publisher 创建了一个数据值。这里 Pulisher 的构造函数 publisher 需要被显式创建，而不能使用默认值构造函数。

【例 4-10】 将上述创建的元组插入到关系表 books 中。

```
INSERT INTO books
VALUES
 ('compilers', array['John', 'White','Black'],NEW Publisher('Springer','Berlin'),
   MULTISET   ['computer','db','ordb'] )
```

说明：实际应用中，可通过指定相应指针（如 author_array[1]）完成对数据的访问和修改。

5. 引用类型

面向对象程序语言提供了引用对象的基本功能，即一个类型的属性值可以是对另一个对象的引用。鉴于此，SQL3 相应提供了一种特殊数据类型——引用（参照）数据类型。在使用“引用类型”时，不是引用对象本身值，而是引用对象标识符 OID。“引用”可分为关于“类型”和关于“元组”的引用。引用类型是和某个特定的其他类型相关联的，其值是相应的 OID。

创建一个类型时，该类型中某个属性可以是对另一个指定类型的引用，如果“属性”是关于指定类型中单个对象的引用，此时语法格式为：

```
属性名 REF(类型名)
```

【例 4-11】 创建一个类型 Class。

```
CREATE TYPE Class AS(name VARCHAR(20),
                     monitor REF(Person) SCOPE People
                    );

CREATE TEBLE class of Class;
```

说明：例中创建的类型有属性 name 和 monitor，而属性 monitor 引用 person 类型。REF(Person)是对 Person 类型中单个对象的引用。另外，“属性”还可以是对指定类型中一

个对象集合的引用，此时语法格式为：

```
属性名 SETOF(REF(类型名))
```

这里，语句中的"SETOF"也可以换为"ARRAY"或"MULTISET"。

在创建关系表时，可以对另一个关系表中元组进行引用，此时需要指明被引用类型的属性所在的元组属于哪一个对象关系表，即此时需要对指向表中元组的引用范围(SCOPE)进行强制性限制，其使用方式与关系模型中的外键类似。限制如果在类型声明中实现，其语法格式为：

```
REF(类型名) SCOPE 表名
```

例 4-11 就是这种情形。

【例 4-12】 定义一个球队类型的语句。

```
CREATE ROW TYPE Class AS(name VARCHAR(20),
                         team_list SETOF(REF(Person))
                         );

CREATE TEBLE class of Class;
```

说明：这里属性 team_list 对类型 Person 的引用就是对 Person 对象集合的引用。

限制也可以在表定义中实现，此时，语法格式为：

```
引用类型的属性名 WITH OPITIONS SCOPE 表名
```

【例 4-13】 将例 4-11 中定义语句改写如下。

```
CREATE TYPE Class AS(name VARCHAR(20),
                     monitor REF(Person)
                    );

CREATE TABLE classes OF Class (monitor WITH OPTIONS SCOPE people);
```

说明：上述两个创建语句中，类型 Class 包含一个 name 属性和一个需要引用类型 Person 的 monitor 属性，而相应的"SCOPE people"或"monitor WITH OPTIONS SCOPE people"将"引用"限制在表 people 中的元组。

4.4.2 继承性创建

通过前面讨论，可以认为"类型"(特别是结构或行类型)和"关系表(类型)"是对象关系数据创建中的"基本"元素。类型之间最基本和常见的语义关联就是继承关系，关系表是行的集合，因此自然需要考虑类型间及关系表间的继承性关联。SQL3 有两种创建继承性的方式，即类型继承性和表级继承性。

1. 类型继承性

类型继承性是指在创建类型时一并创建相关的继承机制。这与 C++ 等类似，表现为类型与类型之间的继承，是在数据模式的层面上创建相关的继承性机制。创建了类型继承性后，由类型语句域约束而产生的所有不同关系表(即关系实例)都具有相应的继承性约束。

类型之间的继承性通过超类型和子类型创建实现。

【例 4-14】 创建 Person、Student 和 Teacher 3 个类型并表示相应的继承关系。

(1) 创建"Person"类型。

```
CREATE TYPE Person(name VARCHAR(10),
                   social_number VARCHAR(18)
                  );
```

如果还需在数据库中存储 Student 和 Teacher 信息，就要创建 Student 和 Teacher 类型。从语义上讲，由于 Student 类型和 Teacher 类型应该是 Person 的子类型，因此，可以通过继承性关系来创建 Student 和 Teacher 类型。

(2) 创建 Student 类型。

```
CREATE TYPE Student UNDER Person(degree VARCHAR(10),
                                 department VARCHAR(20)
                                );
```

(3) 创建 Teacher 类型。

```
CREATE TYPE Teacher UNDER Person(salary INTEGER,
                                 department VARCHAR(20)
                                );
```

说明：上述 Student 和 Teacher 两个类型都继承了 Person 类型属性：name 和 social_number，也分别有各自的属性"degree、department"和"salary、department"。因此，Student 和 Teacher 都是 Person 的子类型，Person 是 Student 和 Teacher 的超类型。其类型层次图如图 4-10 表示。图中箭头方向为超类型，箭尾方向为子类型。

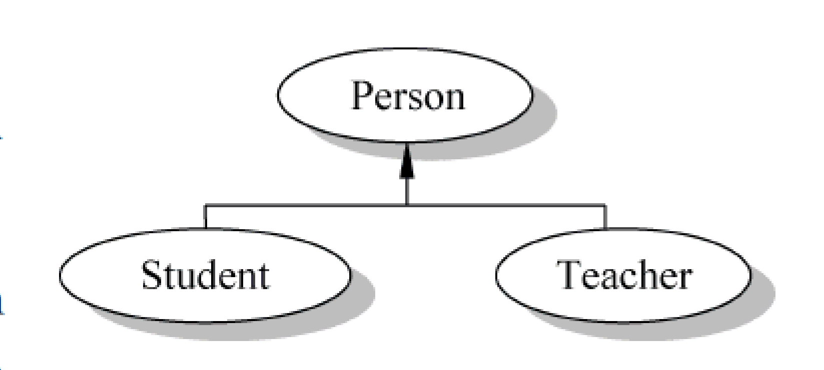


图 4-10 类型级继承

2. 表级继承性

表级继承性是指在创建类型时可不同时创建相关继承性机制，而在创建了关系表即行数据集合时才针对其他已有的关系表而"滞后"创建相应继承性机制。这是在关系实例层面的继承，不同于类型继承情形，同一类型产生的不同关系表有的可以有继承机制，有的可以没有，就是有继承机制的多个关系，也可以具有不同的继承机制，尽管它们都是源自同一个类型约束。从某种意义而言，相对于类型继承性，在实际应用过程中，表级继承性可能具有更多的灵活性与便利性。

对象关系通过子表/超表在表级实现继承性。表级继承性允许将所涉及的关系组成一个类型层次，这是一个有根无环的有向图，此时，子结点可以从一个或多个父结点中继承属性和函数。

1) 子表与超表

在例 4-14 中，定义了 Person 类型之后定义关系 people，其格式为：

```
CREATE TABLE people OF Person;
```

然后分别创建类型 Student 和 Teacher，其格式为：

```
CREATE TYPE Student(degree VARCHAR(10),
                    department VARCHAR(20)
                   );

CREATE TYPE Teacher(salary INTEGER,
                    department VARCHAR(20)
                   );
```

再用继承性创建 students 和 teachers 作为 people 的继承表，其格式为：

```
CREATE TABLE students OF Student UNDER people;
CREATE TABLE teachers OF Teacher UNDER people;
```

说明： people 称为超表，students 和 teachers 称为子表，子表 students 和 teachers 继承了超表 people 的全部属性，其表级继承层次图如图 4-11 所示。

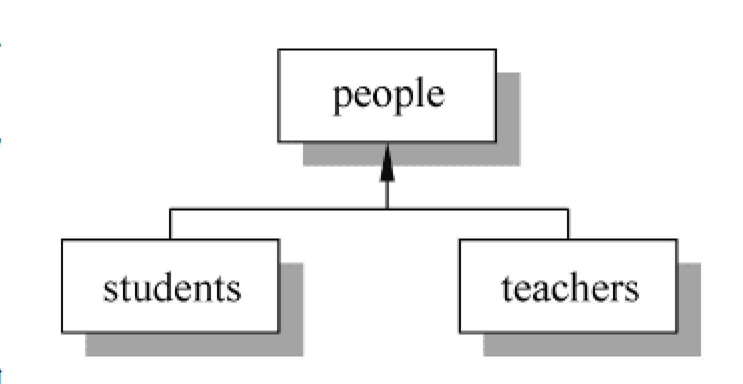


图 4-11　表级继承

2）约束条件

由于定义在表级层面，相比于类型层面就具有了更多的约束性条件，呈现出相对复杂的情形，因此表级继承性需要考虑必要的约束限制。

超表和子表需要满足下述约束性(一致性)条件。

(1) 超表中每个元组需要并且最多只能与每个子表中的一个元组对应。

例如，超表 people 中每个人可以是一个 student，也可以是一个 teacher，可以既是一个 student 又是一个 teacher，也可以既不是一个 student 也不是一个 teacher。

(2) 子表中的每个元组在超表中恰有一个元组与之对应，并且在继承属性上有相同的值。

在上述例子中，如果“子表中的每个元组在超表中恰有一个元组与之对应”不成立，则 students 表中就有可能有两个学生对应 people 表中同一个人；如果“在继承的属性上有相同的值”不成立，students 表中就有可能在 people 表中没有相对应的人。所有这些都与实际情况不符，因此必须避免。

3）表级继承的意义

表级继承可以采取有效的方法存储子表。在子表中不必存放继承得到的属性(超表中主键除外)，这些属性值可以通过基于主键的连接从超表中导出。有了表级继承性概念，对象关系数据模式定义将会更加符合实际，这是因为在没有表级继承的情况下，模式设计者需要通过主键把子表对应的表和超表对应的表联系起来，还需定义表之间的参照完整性约束条件。在表级继承性情况下，可以将在超表上定义的属性和性质用到子表中的对象上，从而可以逐步对 DBS 进行扩充以包含新的类型。

4.4.3　关系表创建

关系数据模型中元组是基本元素，关系表看作同型元组集合；对象关系模型中关系仍然是核心概念，但其基本元素是对象类型，这里对象类型包括行类型(结构类型)和 ADT，“关系”表看作是对象类型的集合。

1. 关系表

如前所述，在数学中将“集合”与“元素”(等价于由该单元素构成的集合)两个不同抽象层面上的概念在同一抽象层面“有意混用”，而在 ORDM 中，也可将“关系”(行类型数据的集合)和“行”(等价于有该行数据构成的集合)在同一层面有意混用，因此就有关系类型之说。实际上，在 SQL3 中，关键词“TYPE”就有意混淆地用来指称“行类型数据”或“关系类型数据”。以下就按照这种考量讨论创建关系表(关系类型数据)的两种基本方式。

(1) 基于关系类型创建。基于关系类型创建即先定义关系模式(关系类型)再定义相应关系实例(关系类型数据)，此时按照“类型外定义行类型”和“类型内定义行类型”细分为以下两种情形。

① 关系类型外定义行类型：在创建的关系类型范围之外定义其中涉及的行类型，此时类型内外行类型名称必须一致。当行类型比较复杂性时，采用此方式可以使得描述更加清晰，校验更为便利。这与 C++在类定义之外具体定义其成员函数相似。

② 关系类型内定义行类型：在创建的关系类型范围内定义其中涉及的行类型，此时，对采用无名称行类型定义的方式(见例 4-15)，这通常用于行类型相对简单或所涉及行类型数量较少的情形。

从本质上而言，此时关系表创建都类似于 RDB 情形。

(2) 按照行类型创建。按照行类型创建即先定义行类型，再将关系类型定义为此行类型数据的集合。

此种方式是 RDB 中所不具有的。

上述关系类型的创建方式如图 4-12 所示。

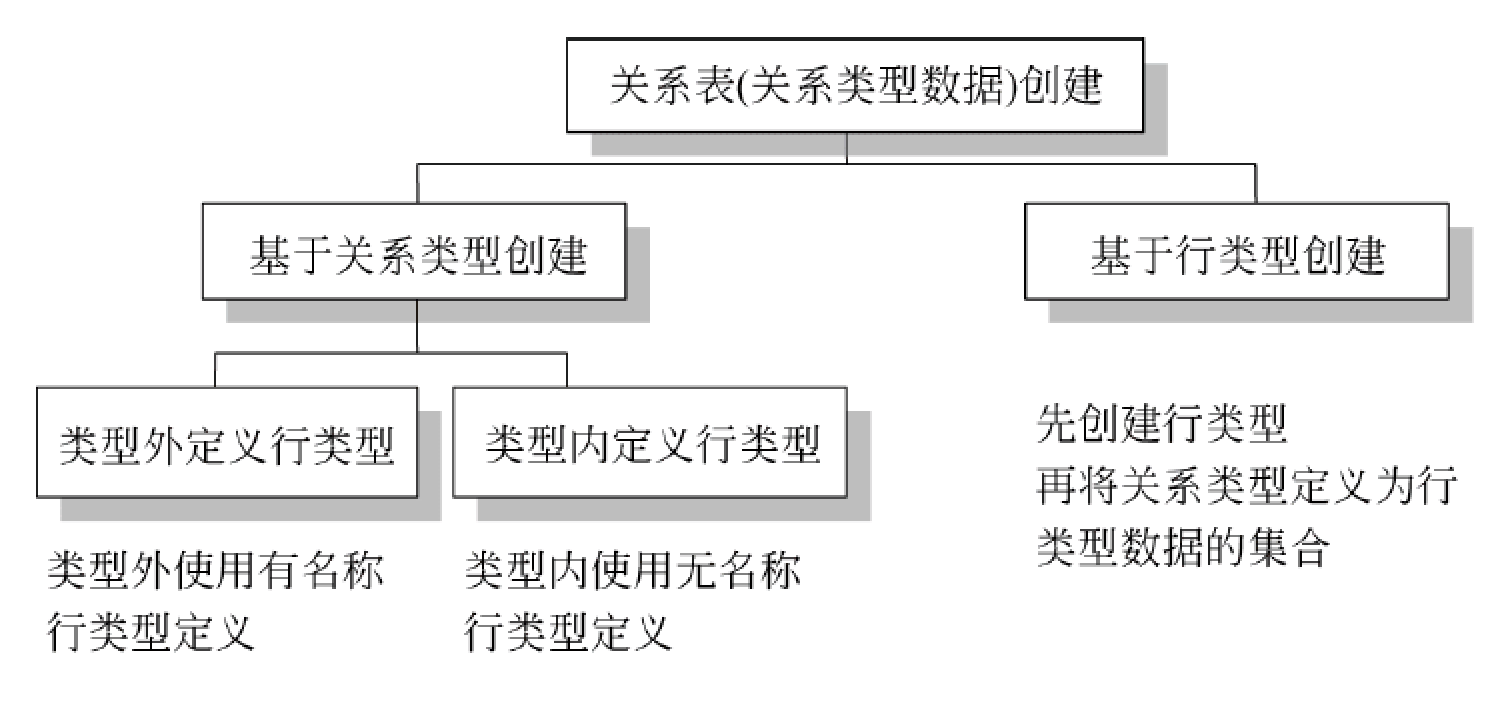


图 4-12　创建对象关系表方式

【例 4-15】 按照上述 3 种方式分别创建大学与教师的对象关系表 universities(uno, unname, city, staff(fno, fname, age))。

1) 基于关系类型创建(类型外定义行类型)

① 先创建“行(结构)”类型 Faculty，其格式为：

```
CREATE TYPE Faculty (fno VARCHAR(10)
                     fname VARCHAR (20),
                     age INTEGER);
```

② 再创建“(嵌套)行”类型 University,其格式为:

```
CREATE TYPE University AS (uno VARCHAR (10),
                           uname VARCHAR (20),
                           city VARCHAR (20),
                           staff SETOF (Faculty)
                          );
```

③ 最后将关系表 universities 定义为行类型 University 的集合,其格式为:

```
CREATE TABLE universities OF University;
```

2) 基于关系类型创建(通过无名称行关系类型内定义行类型)

```
CREATE TABLE university (uno varchar(10),
                         uname VARCHAR (20),
                         city VARCHAR (20),
                         staff ROW (fno VARCHAR (10)
                                    fname VARCHAR (20),
                                    age INTEGER
                                   )
                         );
```

3) 基于行类型创建

① 先创建结构类型 Faculty,其格式为:

```
CREATE TYPE Faculty (fno VARCHAR (10),
                     fname VARCHAR (20),
                     age INTEGER
                     );
```

② 再创建对象关系 universities,其格式为:

```
CREATE TABLE universities (uno VARCHAR (10),
                           uname VARCHAR (20),
                           city VARCHAR (20),
                           staff SETOF (Faculty)
                          );
```

在 ORDB 中,通常行类型都比较复杂,采用直接创建关系类型的方式在某些情况下就难以清楚描述对象关系表的实际构造,一般多采用间接创建关系表即“基于行类型创建”的方式,这实际上是“由里向外”和“由组成部分到整体”的逐层推进定义。

【例 4-16】 设有一个学生选课及成绩的嵌套关系 SC(name,cg(course,grade,date)),其中属性 name、course、grade 和 date 分别表示学生姓名、课程名、成绩和日期。

① 采用基于元组方式创建对象关系表 sc,其格式为:

```
CREATE TYPE CourseGrade (course VARCHAR(20),
                         grade INTEGER,
                         date DATE
                         );

CREATE TYPE StudentGrade AS SETOF(CourseGrade);
CREATE TYPE StudentCourseGrade AS (name VARCHAR(10),
```

```
                                cg StudentGrade
                                );
```

② 在上述基础上再定义关系 sc，其格式为：

```
CREATE TABLEsc OF StudentCourseGrade;
```

2. 综合实例

下面讨论一个按照行类型创建 ORDB 模式的综合实例。

【例 4-17】 定义如图 4-7 所示的对象联系图所表示的数据库。

```
CREATE TYPE Person AS(social_munber VARCHAR (18),
                      name VARCHAR(10),
                      age INTEGER
                     );

CREATE TYPE University AS (uno VARCHAR (10),
                           uname VARCHAR(20),
                           city VARCHAR (20),
                           president REF (Faculty),
                           staff SETOF (REF (Faculty)),
                           edit SETOF (REF (Coursetext))
                          );

CREATE TYPE Faculty UNDER (Person) AS
                          (fno VARCHAR (10),
                           fname VARCHAR (20),
                           age INTEGER,
                           salary INTEGER,
                           work_for REF (University),
                           teach SETOF (REF (Coursetext))
                          );

CREATE TYPE Coursetext AS (cname VARCHAR (20),
                           textname VARCHAR (20),
                           teacher REF (Faculty)
                           editor REF (University)
                               );

CREATE TABLE people OF Person;

CREATE TABLE faculties OF Faculty
                        (works_for WITH OPINIONS SCOPE universities,
                         teach WITH OPINIONS SCOPE coursetexts
                        );

CREATE TABLE universities OF Faculty,
                        (president WITH OPINIONS SCOPE faculties,
                         staff WITH OPINIONS SCOPE faculties,
                         edit WITH OPINIONS SCOPE coursetexts
                        );
```

```
CREATE TABLE courstexts OF Coursetext
                          (teather WITH OPINIONS SCOPE faculties,
                           editor WITH OPINIONS SCOPE universities
                          );
```

说明：需要特别指出，类型"universities"中的 REF(Faculty)和 SETOF(REF(Faculty))中保留词 REF 是不可省的。如果没有 REF 关键词，"faculties"和"universities"这两个表就是递归嵌套，在系统中不可实现。有了 REF 关键词后，相互引用的是关系中元组的标识符(元组地址)，如此就能实现递归结构。本例说明的是通过"引用"机制避免循环嵌套问题，在学习中值得注意。

4.5 对象关系数据操作

使用 SQL3 中的相应语句可实现对象关系数据查询与更新操作。事实上，对传统 SELECT 语句加以适当修改就能处理带有行类型和引用类型的对象关系数据查询。对于数据更新也是如此。此外，在查询语句中，允许用于计算关系的表达式出现在任何关系名可以出现的地方，例如，FROM 子句或 SELECT 子句中，这种可以自由使用子表达式的能力使得充分利用嵌套关系结构成为可能。

4.5.1 数据查询

下面以图 4-7 所示的对象联系图和例 4-17 中定义的 ORDB 为例，介绍 SQL3 中相关数据操作语句的基本使用。

1. 行变量

在一个对象关系表中，需要将其中数据元素——行(元组)看作单独对象，而不看作是以表中的属性作为成员的列表，因此 SQL3 是以关系表中"行(元组)"为操作变量。如前所述，SQL3 中，通常都是有意混用"关系"与"行"概念的，因此可以认为，在传统 SQL 当中，查询语句里实际上是"自然"和"隐式"地将被操作的关系表看作是"行变量"，并将这种行变量直接与属性名联用以求出属性值，但这种机制在 ORDB 中却难以实施，主要原因在于 ORDB 存在嵌套情况，同时属性值具有复杂数据类型，简单地和隐式地将需要操作的关系表当作变量进行求值可能会带来语义上的混乱。所以，在 ORDB 中必须为每个操作(关系)表显式地设置一个"行变量"，然后才可进行有关数据操作。在需要深入到行(对象)内部获取嵌套情况下的有关属性值信息时，可由观察器函数使用所设置的行变量完成相应查询。

【例 4-18】 在例 4-17 的 ORDB 中，查询讲授 DBS 课程中采用 An Introduction to Database System 教材的教师工号和姓名。

```
SELECT F.no, F.fname
FROM faculties AS F
WHERE('DBS', 'An Introduction to Database System') IN F.teach;
```

说明：为对象 faculties 设置行变量 F，然后分别使用观察器函数 F.no，F.fname 和 F.teach。这里实际上省略了观察器函数后面的"()"，如 F.no 实际上是 F.no()等。

【例 4-19】 查询每一位教师开设的课程。

```
SELECT F.no, C.cname
FROM faculties AS F,F.teach AS C;
```

说明：这将新的行 F. teach 设为变量 C。

【例 4-20】 查询广州地区各大学超过 45 岁的教师人数，其格式为：

```
SELECT U.uname,count(SELECT *
                     FROM U.staff AS F
                     WHERE F.age > 45)
FROM universities AS U
WHERE U.city = 'guangzhou';
```

2. 路径表达式

对象关系数据中的嵌套带来数据结构的复杂性。与 RDB 不同，ORDB 为实现有效查询，需要考虑查询的路径表达式。从已知属性值出发搜索未知属性值时经过的属性名称构成的表达式就是路径表达式。路径表达式可以分为下述 3 种情形。

1）属性值为原子值或结构（行）值

当所涉及的属性值均为原子值或结构（行）值时，观察器函数 T. A()中属性引用方式如同常规情形那样，直接在数据对象和相应属性的层次之间添加圆点“. ”来表示路径层次。

【例 4-21】 在例 4-17 的 ORDB 中，查询广州地区的大学校长姓名。

```
SELECT U.uname,U.president.fname
FROM universities AS U
WHERE U.city = 'guangzhou';
```

说明：当属性值为单个值时，只有一个层次点，而当属性值为结构（行）值时，层次点可以逐次延展下去。本例中，为关系表 universities 设置行变量 U，观察器函数 U. president 的返回值仍然是结构值——行，因此校长姓名可用 U. president. fname 表示。

这种由层次点组成，形如 U. president. fname 的表达式就是 SQL3 中的“路径表达式”。

2）属性值为聚集值

当查询过程中存在某个属性值为聚集值时，查询路径表达式就不能接连写下去，此时需要再设置一个新的观察器函数。例如，在某大学里查询教师姓名，不能写成 U. staff. fname，因为属性 staff 通常取多集值而非原子值或行值，否则，此时路径表达式就无意义，需要为 staff 另外定义一个行变量以涉及一个新的查询路径。

在例 4-17 中，为表 universities 设置元组变量为 U，而其分量 U. staff 是一个表（集合值），需要再为其设置为一个行变量“F. ”。

【例 4-22】 查询使用中山大学编写教材的教师工号、姓名和学校。

```
SELECT U.uname, F.fno, F.fname
FROM university AS U, U.staff AS F, F.teach AS C
WHERE C.editor.uname = 'Sun Yat - Sen University';
```

【例 4-23】 例 4-22 中的查询也可以用如下表达形式。

```
SELECT F.works_for. uname, F.fno,F,fname
```

```
FROM faculties AS F, F.teach AS C
WHERE F. works_for. uname = 'Sun Yat - Sen University';
```

3）属性值为引用值

当需要查询属性值为引用类型时，需要表明该属性值的引用身份。

【例 4-24】 查询 Springer 出版社出版的图书名称。

```
SELECT bookname
FROM books
WHERE pulisher_REF.public_name = 'Springer';
```

说明： publisher 的数据类型是引用类型，需要用路径表达式来表示引用对象的属性，如用 pulisher_REF. public_name 表示 pulisher 中的属性 public_name。

【例 4-25】 查询书名为 advanced database 的出版社。

```
SELECT defer(pulisher_REF)
FROM books
WHERE bookname = 'advanced database ';
```

说明： book 中属性 pulisher_REF 是引用类型，而引用类型的值实际上是一个对象标识符 OID。在上述语句中，如果在 SELECT 后直接写 pulisher_REF，则查询结果返回一组 OID 的值，为了得到 OID 所表示的数据实例，使用函数 defer()返回参数所表示数据的具体值。

3. 基于继承的数据查询

SQL3 提供了在继承层面上的数据查询。也就是说，在具有继承出现的场景中，可以只从其中一个关系中查询数据，也可以从该表和它的所有子表中查询数据，这就带来了较大灵活性。实际上，SQL 对此提供了以下 3 种不同的查询语句。以下以实例说明这些查询方式。

设有如下 3 个对象关系类型数据：

```
CREATE TYPE Person(name VARCHAR(10),
                   social_number VARCHAR(18)
                  );
CREATE TYPE Student UNDER Person(degree VARCHAR(10),
                                 department VARCHAR(20)
                                 );
CREATE TYPE Teacher UNDER Person(salary INTEGER,
                                 department VARCHAR(20)
                                 );
```

此时，Student 和 Teacher 都是超类 Person 的子类。

1）只查询超类数据

使用如下语句可只查询超类 Person 中的数据。

```
SELECT *
FROM ONLY Person;
```

此时，查询结果仅为关系 Person 中的所有行类型数据。

2）查询超类和所有子类相关数据

使用如下语句可查询超类 Person 和子类 Student 及 Teacher 中继承于 Person 的数据。

```
SELECT *
FROM Person;
```

此时，查询结果为关系 Person 及子关系 Student 和 Teacher 中继承于 Person 的数据。其中，Person 中行类型结构为“name，social_number”；Student 中行类型结构为“name，social_number，degree，department”；Teacher 中行类型结构为“name，social_number，salary，department”。

对于子关系 Student 和 Teacher 而言，执行此查询语句的结果数据都只有前两个数据项“name，social_number”，从而与 Person 关系得到的查询结果具有相同的数据结构。

3）查询超类和子类所有完整数据

使用如下语句可查询超类 Person 和子类 Student 及 Teacher 中的所有数据与继承 Person 数据所有相关的数据。

```
SELECT Person
FROM Person;
```

此时，子关系 Student 和 Teacher 中的查询结果都具有自身的行数据结构，整体输出结果为具有不同格式结构的 3 个行类型数据集合。

4.5.2 关系与对象关系转换

数据查询本质上可以看作是将一种合法的数据表达式转换为另一种合法的数据表达式。ORDB 应当包含和兼容 RDB，因此需要通过对象关系数据的查询实现同一客观实体的关系形式与对象关系形式之间相互转换。

关系具有原子性，对象关系具有嵌套性，由关系转换为对象关系通常称之为“施加非 1NF”或“施加嵌套”（nesting）；由对象关系转换为关系称之为“解除非 1NF”或“解除嵌套”（unnesting）。在使用 SELECT 语句时，可以要求查询结果以 1NF 形式输出，也可以要求查询结果以非 1NF 形式输出。

【例 4-26】 一个基于对象关系的图书关系表 books 如表 4-1 所示，其中 author_array 和 keyword_set 是聚集类型属性。

表 4-1 图书数据表（非 1NF）

title	author_array	publisher(pub-name，pub-branch)	keyword_set
XML	(John White)	(McGraw_Hill，New York)	(XPath，XQuery)
DB	(White，Smith)	(Springer，Berlin)	(Model，Form)

如果需将该对象关系转换为一个满足 1NF 的平面关系 flat_books，可使用下述语句：

```
SELECT title, A.author, publisher.name AS pub－name,
                publisherbranch AS pub_branch, K.keyword
FROM books AS B, unnest (B.author_array) AS A (author),
            unnest (B.keyword_set) AS K (keyword);
```

说明：上述语句中，FROM 子句中变量 B 被声明为以 books 为取值范围，变量 A 被声明为以 B 的 author_array 中 author 为取值范围。同时，K 被声明为以 B 中 keyword_set 关键字为取值范围。

转换后的平面关系表如表 4-2 所示。

图 4-2　平面关系表 flat_books

title	author	pub_name	pub_branch	keywords
XML	John	McGraw_Hill	New York	XPath
XML	White	McGraw_Hill	New York	XPath
XML	John	McGraw_Hill	New York	XQuery
XML	White	McGraw_Hill	New York	XQuery
DB	White	Springer	Berlin	Data Model
DB	Smith	Springer	Berlin	Data Model
DB	White	Springer	Berlin	Normal Form
DB	Smith	Springer	Berlin	Normal Form

【例 4-27】　将上述平面关系表 flat_books 转换为嵌套关系表。

```
SELECT tilte,auther,Pulisher(pub_name, pub_branch) AS pulisher,
                 COLLECT (keywords) AS keyword_set
FORM flat_books
GROUP BY title, auther, publisher, keyword_set;
```

执行后得到如表 4-3 所示的输出结果，这是一个非 1NF 关系结构。

表 4-3　非 1NF 关系表

title	author	publisher(pub-name,pub-branch)	keyword_set
XML	John	(McGraw_Hill, New York)	(XPath, XQuery)
XML	White	McGraw_Hill,New York	(XPath, XQuery)
DB	White	(Springer,Berlin)	(Model,Form)
DB	Smith	(Springer,Berlin)	(Model, Form)

说明：上述转换为非 1NF 的过程通过对 SQL 分组机制的一个扩充来完成。在 SQL 分组机制的常规使用中，需要对每个组(逻辑上)创建一个临时的多集关系，然后在这个临时关系上应用一个聚集函数。如果不应用聚集函数而只返回这个多重集合，比较容易地完成嵌套转换，创建一个嵌套关系。

【例 4-28】　在前例中，如果还需要将属性嵌套到多集当中，可以执行如下语句：

```
SELECT title,COLLECT(author) AS auther_set
      Pulisher(pub_name,pub_branch),COLLECT (keyword) AS keyword_set
FROM flat_books
GROUP BY title, auther_set,publisher, keyword_set;
```

此语句的查询结果就将平面关系表 flat_books 转换为前述的嵌套关系表 books。

说明：还可以通过查询语句嵌套(使用子查询)来实现 1NF 到非 1NF 的转换。这种基于查询嵌套的方法可以选择使用 ORDER BY 子句将查询结果按照需要顺序排列，从而可

以用于创建数组。

```
SELECT title,
    ARRAY (SELECT author
           FROM authors AS A
           WHERE A.title = B.title
           ORDER by A.position) AS author_array
    Pulisher(pub_name,pub_branch) AS publisher
    MULTISET (SELECT keyword
              FROM keywords AS K
              WHERE K.tilte = B.title) AS kerword_set,
FROM books AS B;
```

上述语句中,关键字 ARRAY 和 MULTISET 说明数组和多集分别应用于查询结构创建。

4.5.3 对象关系数据更新

复杂类型数据可以进行任意层次上的嵌套,由此就使得对象关系数据更新与常规关系数据中情形有所不同。

【例 4-29】 设在对象关系 sc(name,cg(course,grade,date))中某行具下列形式:

```
('Mary',SETOF ('MATHS',95,(1,'July',2005)),('DB',100,(1,'January',2006))));
```

其中,为复杂数据类型的属性 date 创建值的方法是将其各个属性(日、月、年)在圆括号内列出;为集合类型的属性 cg 创建值的方法是在圆括号中列举其中元素并在圆括号前面加关键字 SETOF。

当需要将上面的元组插入到关系 sc 中,可用下面语句实现:

```
INSERT INTO sc
VALUES ('Mary',SETOF ('MSTHS',95,(1,'JULY',2005)),('DB',100,(1,'January',2006))));
```

【例 4-30】 删除在例 4-29 中插入的数据对象。

```
DELETE FROM SC WHERE name = 'Mary'
```

说明: 执行此语句后,SC 中 Mary 的记录全被删除,但其 OID 不能重用。

【例 4-31】 修改 Mary 的 DB 成绩为 98 分。

```
UPDATE sc
SET cg.grade = '98'
WHERE name = 'Mary'and cg.coures = 'DB';
```

说明: 修改后 sc 中相应的 OID 不变。

也可以用通常的 UPDATE 语句完成复合对象关系的更新,这与 1NF 关系的更新非常类似。

本 章 小 结

对象数据库是新一代(第三代)数据库的典型代表。如果以传统 RDB 和关系查询语言 SQL 为基础,扩充关系数据模型到对象关系数据模型,就可建立 ORDB 系统;如果以面向

对象程序设计语言为基础，允许直接在面向对象程序设计语言中使用该语言原有类型访问数据库中的数据，就可建立OODB系统。

1）ORDB本质上是关系结构

从逻辑上考虑，ORDB并不是严格意义下的对象数据库，这是因为它仍然采用关系数据模型，其关键点是扩充了RDB中的数据类型，同时引进了类型的继承机制，但其基础的数据单元仍然是“元组”（其中属性可以取值为复杂类型数据和大对象数据，并且可以进行嵌套表示）和“关系表”（解除了1NF限制并在逻辑和技术上打通关系表与行类型的等价联系）。

但从技术实现上来看，其相应模型SQL3标准已经收入了相当多的面向对象内容，为适应对象概念提供一个高级接口，从而打通从关系的“抽象世界”到面向对象的“真实世界”的一条前景广阔的路途。

（1）扩充传统关系数据类型。将传统关系数据类型扩充到行类型和聚集类型，由此就自然引入扩展数据类型对象间新的关系——嵌套关系和引用关系；由于引用关系等因素考量，还可以对基本对象赋予对象标识。

（2）引入基本对象类型。SQL3通过引入复杂数据类型解决了RDB中类型简单不足以适应新的数据管理需求的问题，但这些复杂类型可能需要适合于自身计算的特定函数并且实现技术封装，由此还需要引入ADT。ADT类型提供了必要的数据操作机制，同时还可以由SQL3提供的类型构造器生成更为复杂的用户类型，因此，ADT类似于面向对象方法中“类”概念。需要注意的是，按照SQL3标准，只将行数据类型和抽象数据类型看作对象类型（类），而内置数据类型、聚集和引用数据类型通常只看作是基本数据类型，类型是简单的基本类。

SQL3中数据基本数据类型和对象类型如图4-13所示。

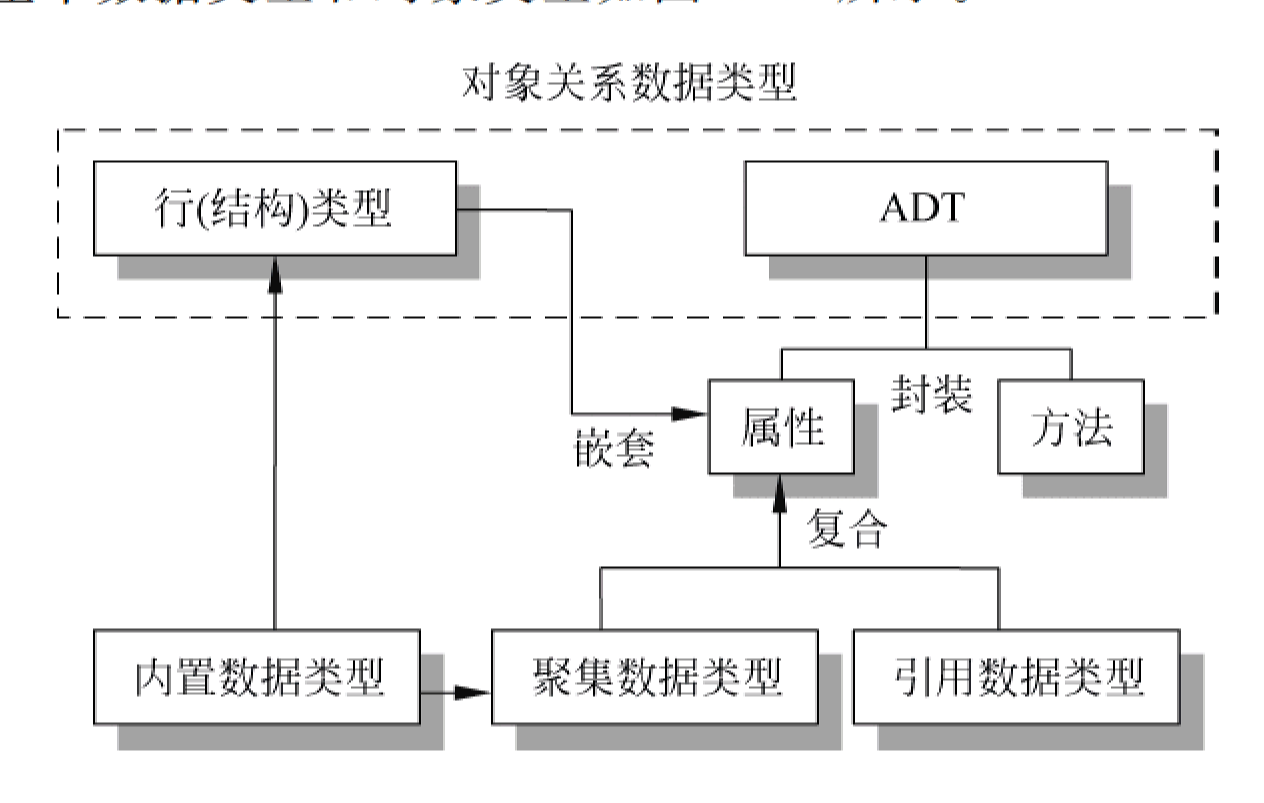


图4-13　SQL3中数据类型与对象类型

（3）类型继承和子表继承。借鉴面向对象方法中的对象和类型概念，引入类型之间的继承关系，解决了类的细化与泛化问题。

（4）元组变量与查询路径。在SQL3中，查询具有两个显著特点：一是查询子句可以嵌套在SELECT语句中任何可使用关系名称的地方；二是数据操作的基本单元是元组。因此，需要每个关系表设置一个元组变量。如果需要查询元组中的属性值，则需要调用内置函数——观察器函数。由于嵌套的原因，当属性值为原子值或结构值（元组值）时，可将层次点逐次写下去，形成查询路径；当属性为集合值时，就需要再设置元组变量，进而再调用观察器函数。

2）ORDB 的实现

作为 RDB 的面向对象扩充，现有 ORDB 大多都是对已有 RDB 进行技术层面上的扩展。为了使对存储系统中关系存储和索引等的改动最小化，通常是将 ORDB 的复杂数据类型转化为 RDB 的简单数据类型。由于 E-R 模型中的情形在 ORDM 中基本得到完全实现，因此这种转换可以借鉴 E-R 模型到关系模型转换而进行开发设计。当然，这会带来较多的数据冗余，需要有更多更精细的技术处理。这种 ORDB 的实现方式通常称为“松耦合”方式，即以 RDBMS 为内核，在其外层整合一个面向对象层，如图 4-14 所示。

图 4-14 松耦合方式

松耦合方式以传统关系表为基本数据结构，在对象外层增加了复杂数据类型(聚集和结构类型)、引用和继承机制。当前大多数 ORDBS 都采用这种结构，如 Oracle 的 ORDBNS 等。

此外，还有将关系与对象关系功能相互融合紧密整合的“紧耦合”方式，此种方式效果更好，但开发研制的难度也更大，如 Infomix 的 Unidata 就是如此。

3）ORDB 与 OODB 比较

ORDB 与 OODB 的比较如表 4-4 所示。

表 4-4 ORDB 与 OODB 比较

对象关系数据库(ORDB)	面向对象数据库(OODB)
从 SQL 出发，引入复合数据类型、引用数据类型和继承性概念，形成 SQL3 标准	从 OOPL 和 C++出发，引入持久性数据概念，对数据库进行操作，形成持久化 C++系统
SQL3 数据查询语言 DDL	ODMG ODL(与 DDL 差异较大)
SQL3 数据查询语言	ODMG OQL(具有 SELECT 语句风格)
具有结构化和非过程性查询特征	具有导航式和非过程性查询特征
符合第四代语言	符合面向对象语言
隐式联系	显示联系
主键概念，对象标识概念	唯一对象标识符
能够表示“对象”	能够表示“关系”
关系是核心概念	对象是核心概念

主要参考文献

[1] 肖海蓉，任民宏. 数据库原理与应用[M]. 北京：清华大学出版社，2016.

[2] Abraham Silberschatz，Henry F Korth，S Sudarshan. 数据库系统概念[M]. 5 版. 杨冬青，唐世渭，等译. 北京：机械工业出版社，2007.

[3] 鞠时光. 对象关系型数据库管理系统的开发技术[M]. 北京：科学出版社，2001.

[4] 徐洁磐. 面向对象数据库系统及其应用[M]. 北京：科学出版社，2003.

[5] 施伯乐，丁宝康. 数据库技术[M]. 北京：科学出版社，2002.

[6] 丁宝康，董健全. 数据库实用教程[M]. 北京：清华大学出版社，2001.

第5章 空间数据库

空间数据库(Spatial Database,SDB)主要用于存储管理空间数据(如地理信息数据)和非空间数据(如关于空间对象说明信息数据)。SDB 在相应的查询语言中提供各类空间数据类型,能够进行空间索引,并具有完成各类空间数据查询和空间关系分析的有效机制。数据模型是数据库研制的基础;数据结构是数据模型的决定性因素;数据之间相互关系是数据结构赖以生成和得以抽象的基础。作为一种特定的数据管理技术,SDB 中的数据对象与 RDB 中的数据对象的区别在于它包含“空间”形体的几何形状和所处位置,因此空间数据对象之间相互关系具有更为复杂的语义描述。本章首先对 SDB 进行简要介绍,然后描述空间数据库模型、空间索引及空间数据库的系统结构。SDB 来源于地理信息系统(GIS)提出的数据管理需求,同时也是该系统的核心技术,本章中的相关原理和技术多围绕 GIS 的典型应用实例展开讨论。

5.1 空间和空间数据

空间和时间是客观万物赖以存在与活动的基本框架,数据是客观世界中各类实体相应特征的反映与表现,存储和管理空间数据是数据库基本的技术领域之一。

5.1.1 空间与空间实体

同时间概念相比,空间概念相对比较成熟,根据不同学科空间概念使用的基本需求,对空间有着不同的理解和界定。

(1) 数学。数学中的空间是指具有一定结构和对相应运算封闭的数学对象集合,这些集合在特定环境中具有“直观”空间的意义和想象。事实上,数学空间是具有抽象结构关系和丰富研究内容的特定集合。

(2) 经典物理学。牛顿经典力学中空间是指物体在 3 个相互垂直方向上所具有的广延性,是物体存在的环境和发生相互作用的背景。

(3) 现代物理学。爱因斯坦相对论力学中空间作为时空连续体的一个组成部分而存在,空间和时间可以相互转换,彼此关联,不可分离。

(4) 地理信息系统。GIS 中的空间概念基于经典物理学框架,基本含义是客观对象(物质、能量和信息)在形态、结构和功能关系上的分布方式及其在时间上的延续。

SDB 起源于 GIS 数据管理,其原理讨论的基础和技术展开的支撑是经典物理学中的空间概念,并以 GIS 为重要应用背景。GIS 中的空间具有绝对空间和相对空间两种形式。

(1) 绝对空间:具有用空间位置属性进行描述的“空间位置”的集合,由一系列不同位

置的空间坐标值所组成。

（2）相对空间：具有空间关系属性特征的"空间实体"集合，由不同实体之间的空间关系构成。

GIS 中的地理空间被建模为二维或三维欧氏空间中的子集，在其中存在的客观实体（地理实体）称为空间实体或空间对象。空间实体通常被建模为结点、弧段、多边形及基于其上的其他"组合"型的空间形体。空间实体基本要素是其形状、位置和相互之间的空间关系。根据所讨论空间实体间关系的不同可以定义不同的空间类型，如距离关系对应度量空间中的数据类型，拓扑关系对应一般空间中的数据类型等。

在给定空间中，SDB 主要关注空间实体对象的下述基本特征。

（1）维数特征。空间和非空间系统基本区别是"多维性"，即相应系统支持的空间维度。一般而言，n 维系统支持的空间维数小于或等于 n。在欧氏空间中通过独立参考轴数量确定维数。在 GIS 中，二维和三维空间通常是指"纯粹"的二维或三维欧氏空间，而四维空间通常是指三维空间和一维时间的整合体。从二维到 n 维从数学上来看应当没有本质困难，但从计算机技术实现上考虑却有相当大的差异。当前 SDB 研究的空间实体多是针对二维空间。

（2）位置特征。研究空间对象首先需要对其进行定位，此时应该存在一个参照系统用以描述对象的绝对或相对位置。欧氏空间中一般采用笛卡儿坐标系或极坐标系作为通用参考系统。

（3）几何特征。空间对象具有各种各样几何形状，人们对于空间对象最直接和突出的认知就是其所具有的几何形状。空间对象分为人工对象和自然对象，前者主要指建筑物、道路、管道和通信网络等；后者主要指山脉、河流和湖泊等。但不论是人工对象还是自然对象，其集合特征通常都可以抽象为点、线、面和体等几何形体进行讨论研究。

（4）关系特征。为了建立数据模型，需要研讨相关数据对象相互间的关系。空间对象不仅其形状极其复杂，而且相互间具有基于距离、方位和拓扑等多层次的关联，如两个空间对象距离远近、一个在另一个的东南面、一个和另一个有无相交、相邻等关系。这些关系一般都具有描述刻画繁复、技术处理困难等特点，是 SDB 研发过程中的关键问题之一。

5.1.2 空间数据

随着计算机技术发展和应用领域扩大，数据库不仅要处理"点"数据，还需处理非点数据，其中就包括描述一维、二维和三维空间对象的数据。

1. 基本概念与特征

空间数据是指以一维、二维和三维空间实体为描述对象的计算机意义下的数据。

空间数据早期应用主要来自计算机辅助设计和几何应用，现已扩展到机器人、计算机视觉和图像识别等方面，特别是地理信息处理为空间数据管理技术研究开发提供了巨大应用需求与发展动力。空间数据作为一种非常规数据具有如下特征。

（1）数据量巨大。空间对象可以相对简单，如点、线等；也可以极其复杂，如地形图和资源分布图等。即使是简单对象，相应数据量也往往巨大，而复杂对象通常需要分成相当精细的栅格描述，其数据量更是可观，单个对象数据就常常达到兆字节以上。

（2）数据结构复杂。空间数据需要描述具有复杂结构或嵌套层次的对象，如不规则的

多维空间物体和分子结构等。这些对象本身的结构表示要比常规数据对象更为精致复杂；同时，空间对象通常是二维以上图形，相互间关系呈现出多样化的特征且多与应用有关，如相交、平行、邻近、包含、覆盖和相切等。

(3) 查询方式多样。空间数据一般按照其空间特征和与其他空间数据的相互关系进行查询。空间对象形状通常不规则，查询条件的描述和验证相当不易，如查询两个不规则区域相交的面积就是一项较为困难的工作。当空间对象是三维时，如多面体，计算它们的相交空间更是极其复杂。

(4) 难以确定顺序。对于一个空间如平面区域集合，其中元素形状大小不同，彼此之间可以包含、重叠、覆盖和不相交，因此难以为其定义一个空间次序，以致于建立在一维数据排序基础上的有效算法，如排序和归并算法等，都不能有效用于空间数据；即使定义出某种空间顺序，也可能由于该顺序的过于“人为化”而丢失原有信息，进而缺少有效的实用价值。

2. 空间数据语义范畴

处理数据的关键在于表达确定的相关语义信息。作为一种异于常规情形的计算机数据，空间数据需要包括下述 3 个层面的基本语义。

(1) 空间语义：主要是空间定位、相关度量和相互关系信息。

① 空间定位：确定事物出现在何处及事件发生于何地。

② 空间度量：用于计算物体的长度、面积和体积，以及物体之间距离与相对方位。

③ 空间关系：用于确定物体之间的分布和拓扑(如邻接、关联和包含等)关系。

(2) 非空间语义。主要是空间数据的属性信息，分为专题属性和质量描述两种情形。

① 专题属性：对空间物体进行语义描述，表明该物体是什么，如“一座具有产权的商住楼”。

② 质量描述：主要说明某些具有补充性质的信息，如质量、数量和等级等。

(3) 时间语义：空间和时间相互联系不可分割，描述物体空间语义时通常也需研究物体动态变动即物体要素的时序变化，实现对空间数据更新、对历史数据积累和对未来改变的预测。

在空间数据管理领域内，主要研究(1)和(2)两种语义信息。由于空间信息和非空间信息存在较大差异，如何进行两者有效存储与整合处理是 SDB 中的一项基本技术。当前空间信息与非空间信息在存储与连接方面主要有下述几种方式。

(1) 不同系统分别管理。通过彼此独立的数据库系统对空间与非空间信息进行分别存储管理，使用对象标识技术建立两者之间的对应关系。

(2) 同一系统统一管理。扩充常规数据库系统的空间数据管理功能，在同一系统中存储管理空间与非空间信息，使得两者具有密切联系，从而有效利用常规系统的现有功能，但这也需要使空间信息表示适应于常规数据模型，降低了系统运行效率。

(3) 统一结构管理。建立空间与非空间信息的统一结构，建立双向指针参照，统一由一个数据库管理系统进行控制，从而提高系统的灵活性和应用范围。

5.2 空间数据模型

与其他数据模型相比，空间数据模型一个突出的特点就是其概念的提出和引入与相应的实际应用密切相关。空间数据管理起源和发展于 GIS 的建立和推动，以 GIS 为应用背景

讨论空间数据模型就成为一种自然和有效的途径。

5.2.1 数据类型与数据模型

数据类型本质上和数据模型相通，也是研究数据模型的基础。以下讨论 GIS 中二维空间数据类型为主，相应结果可以推广到三维和三维以上的情形。

1. 空间数据类型

在 GIS 中，基本空间数据类型主要有下述几种情形。

(1) 点(point)：仅有空间位置而不表示其范围(Extent)的对象，如 GIS 中的城市等。

(2) 线(line)：具有其上多个点的空间位置描述同时还具有其在空间延伸范围及长度的空间对象，如河流、道路、管道、航线、等高线和等降雨线等。

(3) 区域(region)：具有空间位置及面积和周长等参数以表示其覆盖范围的空间对象，如森林、湖泊和行政区域等。

以上述基本空间数据类型为基础可建立如下两种“导出”的空间数据类型。

(4) 划分(partition)：将一个区域按其自然、行政或其他特征分解成若干子区域，当这些子区域互不相交同时其“并”又覆盖该区域时，则此子区域集合就是该区域的一个划分。例如，国家行政区域划分图和土地利用图等。划分可嵌套，如国家分成省市、省市分成县区和县区分成乡镇等。

(5) 网络(network)：由若干点和一些点与点之间的连线组成图形结构，如公路网、河网、电力网、电话网和交通线路图等。

2. 空间数据模型

空间数据管理涵盖的应用范围极其广阔，不同需求有着不同的数据应用模式，从理论上来讲，需要有一个共同的数据模型作为基础以便进行系统的整体构建设计，同时也作为进入系统的空间数据正确与否的判断及进行各类应用系统比较评价的规范标准。然而，由于空间数据自身结构形状极为复杂，描述和处理相互之间空间关系更为困难，因此在技术实现上难以像关系数据库那样提出一个统一模型的框架思路，只能依照对现实世界提取空间数据的不同视角和方式建立相应的空间数据模型。如前所述，在 GIS 中，有基于空间坐标(主要是空间位置)的绝对空间和基于属性特征(主要是空间关系)的相对空间之分，通常就以此建立相应的两种不同空间数据模型，即基于绝对空间概念的矢量模型和基于相对空间概念的镶嵌模型。

1) 矢量模型

矢量模型(vector model)是将空间对象抽象为确定的可识别的数据对象，并使用点、线、面和其他几何体等一组基本空间数据类型描述空间对象。

在矢量模型中，前述基本空间数据类型可以表示如下。

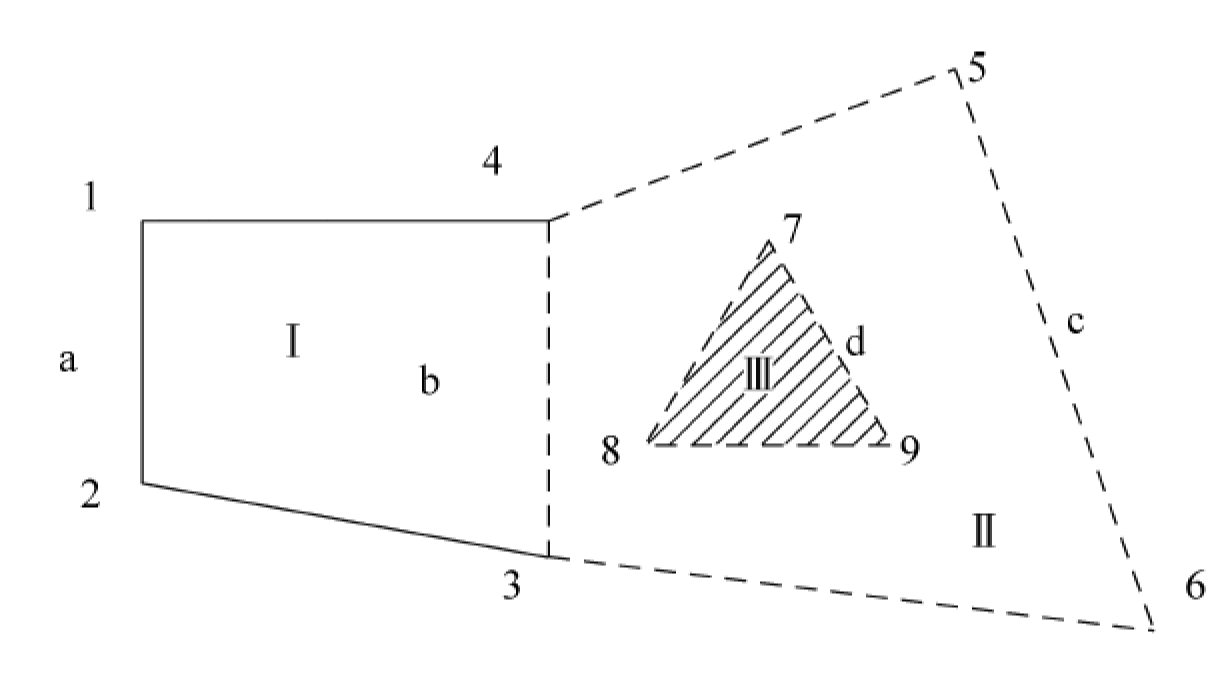
(1) 点对象：由单独坐标对(x,y)表示。例如，使用点描述大比例地图上的城市等零维对象。

(2) 线对象：由坐标对序列(x_1,y_1)、(x_2,y_2)、…、(x_n,y_n)表示。例如，使用线描述河流、铁路和公路线等一维对象。

(3) 面对象：由一组有序且首尾相接的坐标集合$\{(x,y)\}$(即线对象)表示其轮廓范围，即(x_1,y_1)、(x_2,y_2)、…、(x_n,y_n)。例如，使用面描述各个行政区域等二维对象形状。

（4）复杂体对象：由一般几何体对象即点、线和面的集合体表示。例如，由一般几何体描述建筑的集合、群岛等更为复杂的空间对象。

一个空间对象实例的基于矢量模型的数据抽象如图 5-1 所示。



(a) 绝对空间中的空间实体

1	2	3	4	5	6	7	8	9
x_1, y_1	x_2, y_2	x_3, y_3	x_4, y_4	x_5, y_5	x_6, y_6	x_7, y_7	x_8, y_8	x_9, y_9

(b) 点对象的数据建模

线编号	始点	终点	点编号
a	4	3	4，1，2，3
b	3	4	3，4
c	4	3	4，5，6，3
d	7	8	7，9，8

(c) 线对象的数据建模

多边形编号	边界编号
Ⅰ	a，b
Ⅱ	d
Ⅲ	b，c

(d) 面对象的数据建模

图 5-1　基于矢量模型的空间对象建模

矢量模型按照对空间坐标数据组织与存储方式的不同分为下述两种情形。

（1）拓扑模型：将空间实体间某些拓扑关系直接进行存储而没有数据冗余。

（2）非拓扑模型：空间对象用一系列坐标串表示。点对应于坐标序对(x,y)，线对应于坐标序对(x,y)的序列，面对应由起点和终点坐标表示的线围成的多边形。由于模型记录空间实体形体信息而没考虑空间实体间相互关系，此时需要通过在数据文件中搜索所有实体信息并经过大量计算得到对象的空间关系，按照此模型难以有效进行空间分析，通常只适用于无须考察对象空间关系的领域（如地图制图等）。

矢量模型能够方便地表示空间对象之间的拓扑关系，图形精度较高而数据存储量较小，易于定义和操纵单个数据目标，还可直接进行坐标变换和距离计算等操作，但缺乏与遥感及数字地面模型直接结合的能力，数据结构比较复杂，难以进行重叠等操作。

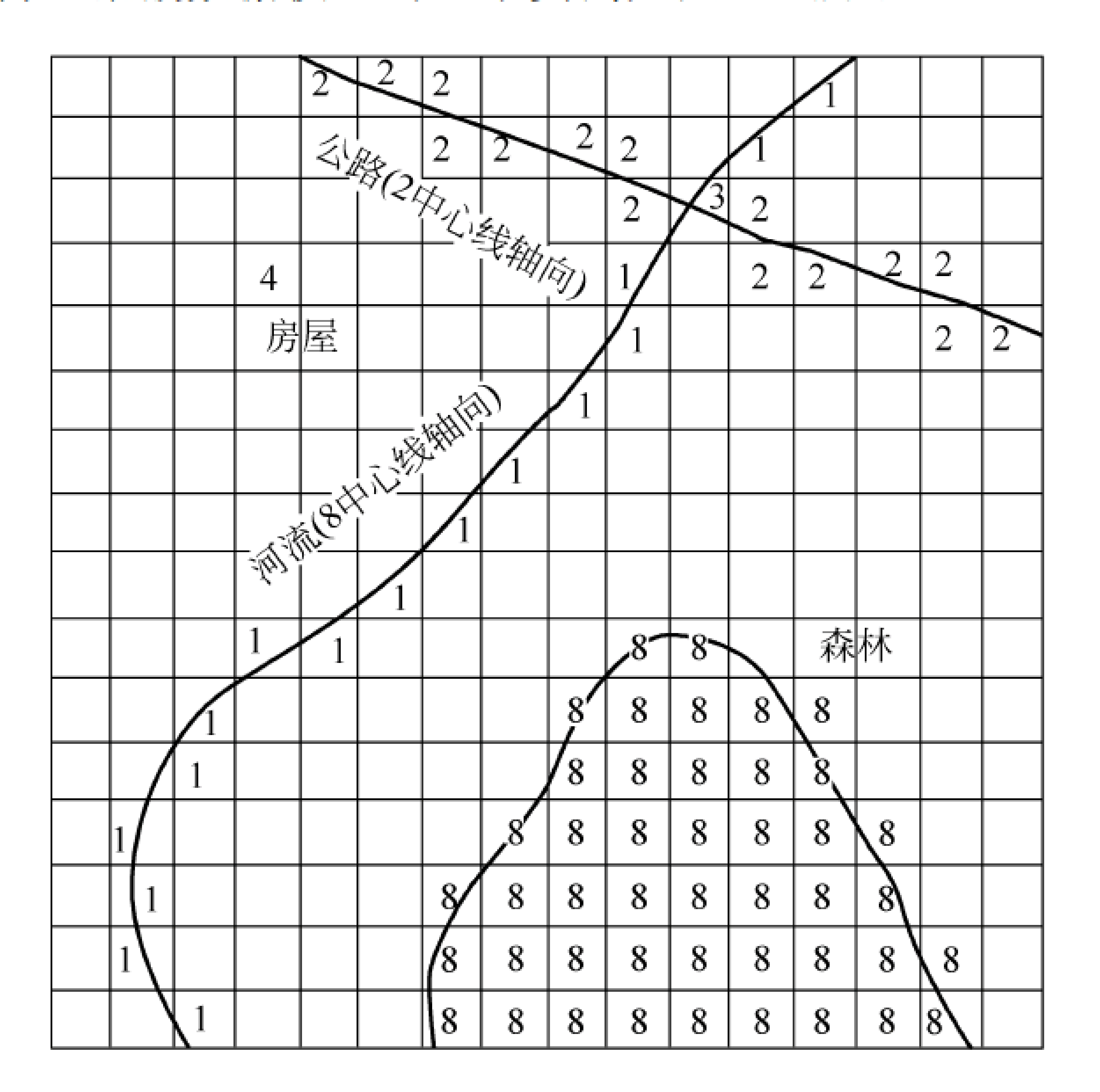
2）镶嵌模型

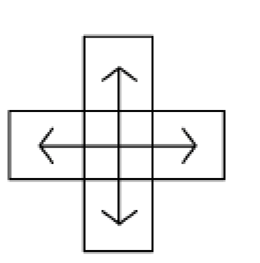
镶嵌模型（tessellating model）将所考虑的空间对象所处的平面块（如地球表面）划分为

大小相等的网格阵列，每个网格作为一个像素或基元(voxel)。网格中的所有像素按照行与列进行定义，各个像素相互邻接、自身连通和互不重叠。网格行列中的每个像素包含一个代码表示其属性类型与取值，也可以仅包含指向其属性记录的指针。

作为一种简单直观的空间数据结构，镶嵌模型也称为网格结构或像素结构，或者称为基于域的栅格数据模型。

由上述分析可知，镶嵌模型中的栅格数据是以规则的阵列表示空间实体或现象分布的一种数据结构。镶嵌数据模型的一个实例如图 5-2 所示。





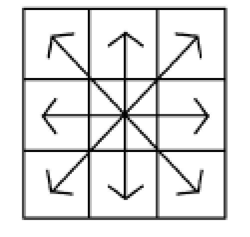


图 5-2　镶嵌数据模型的一个实例

在镶嵌模型中，前述基本空间数据类型可以表示如下。

(1) 点对象：由包含其在内的单独像素表示。

(2) 线对象：由其中心轴线上的像素序列表示。

(3) 面对象：由覆盖其上的像素集合来表示。

按照所取像素或基元相同与否，镶嵌模型可分为下述两种情形。

(1) 规则镶嵌：基元彼此相同，通常由规则正方形、三角形等组成。常用方法有栅格矩阵、行程编码和四叉树等。

(2) 不规则镶嵌：基元多边形随着地形变化而变动，常用方法有不规则三角网、泰森多边形、Voronoi 图和 Peano 曲线等。

镶嵌模型可直接利用遥感及数字摄影测量和扫描等途径获得栅格形式数据，数据结构简单，计算像素和其邻域间方位、邻接即连通等拓扑性质比较方便，可以有效实现地图或图像重叠位置运算并进行相关空间分析。但当数据精度增加时，数据量会不断增长，对系统存储要求较高，同时查询速度也会降低，难以实现空间实体的旋转及坐标变换操作，对空间实体识别与标识操作也相对困难，需通过赋予基元不同标识号来区别不同空间实体。

5.2.2 空间对象关系

数据模型的基本点在于数据对象的表示和数据关系的描述。空间数据对象的表示就是其空间位置和空间形状的表示，而数据相互关系描述就是其度量关系、拓扑关系和位置方位关系的描述。相对于常规数据，空间数据在这两个方面都具有自身显著特点，同时也构成空间数据模型讨论的基本内容。从数据管理角度考虑，空间数据表示主要通过空间近似图形实现，而空间数据关系描述则主要通过度量计算和拓扑刻画实现。

如前所述，空间数据的“空间”是“欧氏空间”。下面给出空间数据基本平台欧氏空间的数学概念。

欧氏空间：设 R 为实数域，V 是 R 上向量的非空集合，若在 V 上定义满足如下条件的称为内积的一个二元函数 $<x,y>$，则称 V 为 R 的欧氏空间。

(1) 非负性：$<x,x>\geqslant 0$，$<x,x>=0 \Leftrightarrow x=0$，$x\in V$。

(2) 对称性：$<x,y>=<y,x>$。

(3) 线性性：$<\alpha x+\beta y,z>=\alpha<x,z>+\beta<y,z>$，$\alpha,\beta\in R$；$x,y,z\in V$。

直线 R，平面 R_2 和空间 R_3 通过适当定义内积之后都是欧氏空间。

1. 空间关系类型

GIS 在欧氏空间中讨论其中空间对象相互间的关系。这些关系可以分为基于度量、拓扑和方位的 3 种类型。

1) 度量关系

设有一个集合 E，如果在 E 上定义了一个二元函数 $d(x,y)$，$x,y\in E$，满足如下条件。

(1) 非负性：$d(x,y)\geqslant 0$。

(2) 对称性：$d(x,y)=d(y,x)$。

(3) 三角不等性：$d(x,y)\leqslant d(x,z)+d(z,y)$。

则称 E 是一个度量空间，$d(x,y)$ 称为 E 上的度量函数。

欧氏空间就是最常用的度量空间。作为度量空间，欧氏空间中坐标轴上两点之间的距离函数——绝对值函数，以及二维平面上和三维空间中的距离函数——相应坐标差完全平方之和的方根函数等都是满足上述 3 条的度量函数，一般度量函数可看作是通常距离函数的推广。

基于度量函数的空间对象度量关系主要有下述情形。

(1) 两个点对象之间的距离。

(2) 单个线对象的长度。

(3) 单个平面图形的面积和其边界曲线的长度。

(4) 单个空间对象的体积和其表面积。

(5) 一个空间对象相对于另一个空间对象的距离。

2) 拓扑关系

拓扑关系是空间对象间最重要的一种相互关系。

基于拓扑的空间对象关系主要有下述情形。

(1) 邻接(meet)：两个空间对象之间具有共同的边界。

(2) 包含(contain)：一个空间对象作为点对象的集合包含在另一空间对象的点集合

当中。

(3) 交叠(overlap):一个空间对象看作点的集合与另一空间对象的点集合相交。

上述三类是空间数据查询中最常出现的拓扑关系。例如,行政区域间的相邻,城市包含其中的公园,发生地震区域与发生火灾地域的重叠等。空间拓扑关系是空间分析的基本工具,空间数据特征在很大程度上取决于数据拓扑性质的合适描述与有效存储。空间拓扑关系表示与刻画需要采用数学中点集拓扑和代数拓扑的基本原理和方法。

基于拓扑关系的空间数据查询通常需要考虑下述两个问题。

(1) 查询所有与给定对象具有某种拓扑关系 R 的空间对象。

(2) 给定对象 A 和 B 具有怎样的拓扑关系。

为了从数学上描述空间对象拓扑关系,需要分析空间对象的组成成分。

设 A 是空间对象,引入下述概念与符号。

① A 的内部 A°:A 的所有"内点"的集合(如果一个点 a 存在一个邻域包含于 A 内,则称 a 为 A 的内点)。

② A 的外部 A^{-}:A 的所有"外点"的集合(如果一个点 a 存在一个邻域包含于 A 的补集 A^{c}内,则称 a 为 A 的外点)。

③ A 的边界∂A:A 的所有"边界点"的集合(如果一个点 a 存在一个邻域同时与 A 和 A^{c}内相交,则称 a 为 A 的边界点)。

两空间对象 A 和 B 之间的二元拓扑关系基于各自对象成分的相交(Intersection)关系。

空间对象 A 和 B 的 6 个组成部分分别构成下述 9 种相交情况:$A^{\circ}\cap B$、$A^{\circ}\cap\partial B$、$A^{\circ}\cap B^{-}$、$\partial A\cap B^{\circ}$、$\partial A\cap\partial B$、$\partial A\cap B^{-}$、$A^{-}\cap B^{\circ}$、$A^{-}\cap\partial B$、$A^{-}\cap B-$。

考虑到布尔判断中的取值情况为{0,1},可以确定共有 $2^{9}=512$ 种二元拓扑关系。在实际问题当中主要研究其中的 8 种彼此互斥关系:相离(disjoint)、邻接(meet)、交叠(overlap)、相等(equal)、包含(contain)、在内部(inside)、覆盖(cover)、被覆盖(covered by)。

3) 方位关系

与前述"度量"和"拓扑"关系不同,空间对象方位关系没有相应的数学工具支撑,呈现比较复杂的情形,通常使用"上下""左右"和"东南西北"等方向概念描述方位关系。

在实际应用中,空间对象方位关系分为下述 3 种情形。

(1) 绝对方位:全球定位系统背景下的方位,如东、西、南、北、东南、西南和东北等。

(2) 相对方位:根据与给定目标的方向来定义的方位,如左右、前后和上下等。

(3) 基于观察者方位:按照专门指定的称为观察者参照对象来定义的方位。

2. 空间关系谓词形式

由于"度量"和"拓扑"关系本质上都是数学关系,因此可以方便地进行相应的谓词形式描述,从而为在计算机实现这些关系提供原理支持。为讨论空间关系谓词描述,需要事先设定相应的基本符号。

(1) SDT:空间数据类型。

(2) ZS:大小为零(zero size)的空间数据类型,如点。

(3) NZS:大小非零(non-zero size)的空间数据类型,如线、区域等。

(4) ADT:原子(atomic)空间数据类型,如点、线、区域。

(5) CDT：集合型(collection)空间数据类型，如网络、划分等。

(6) PT：点。

(7) LN：线。

(8) RG：区域。

(9) PTN：划分。

(10) NTW：网络。

上述空间数据类型对象的层次关系，如图5-3所示。

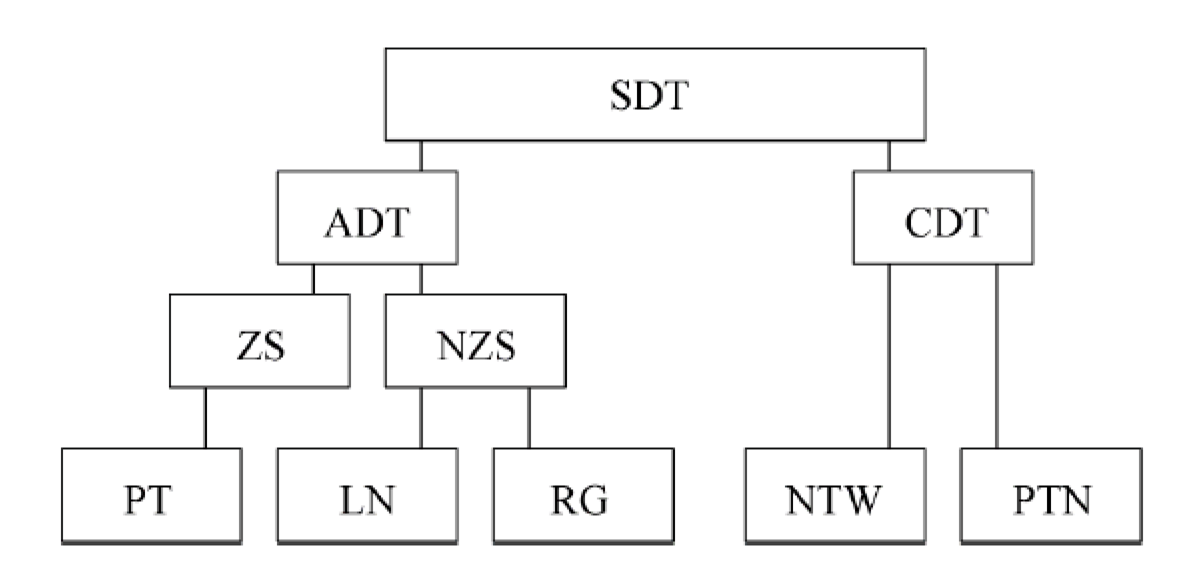


图5-3　空间数据类型对象的层次关系

1) 度量关系描述

(1) 两点间距离(DIST)：PT×PT→NUM DIST。

(2) 两空间图形间的最大、最小距离(MAXDIST，MINDIST)；SDT×SDT→NUM MAXDIST或MINDIST。

(3) 多点的直径(DIAMETER)：PT→ NUM DIAMETER。

(4) 线的长度(LENGTH)：LN→ NUM LENGTH。

(5) 区域的周长(PERIMETER)或面积(AREA)：RG→ NUM PERIMETER或AREA。

2) 拓扑关系描述

(1) 两个同类型空间数据是否相等(＝或≠)：

PT×PT →Bool；

LN×LN→ Bool；

RG×RG → Bool。

(2) 空间数据SDT是否在区域RG中(INSERT)：

SDT× RG →Bool。

(3) 两非零空间数据是否相交(INTERSECTS)：

NZS× NSZ→ Bool。

非零空间数据相交有下述3种情形。

(1)条线相交为点的集合：LN×LN→2^{PT}。

(2)线与区域相交为线的集合：LN×RG→2^{LN}。

(3) 区域与区域相交为区域的集合：RG×RG→2^{RG}。

其中，如果设E是集合，那么2^E表示E的幂集。

上述空间对象相交情形，如图5-4所示。

(4) 两区域是否邻接(IS—NEIGHBOR—OF)：

RG×RG→Bool。

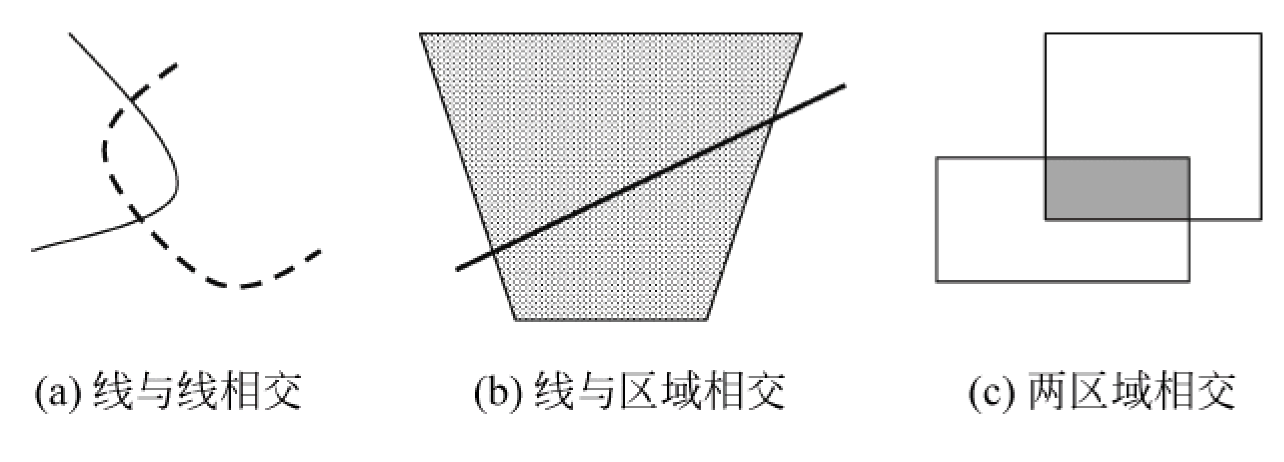


图 5-4　空间对象相交情形

两个区域邻接情形如图 5-5 所示。

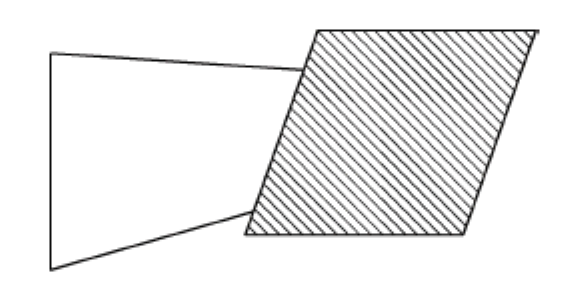


图 5-5　两个区域邻接情形

(5) 重叠(OVERLAP)

PTN×PTN→2^{FG}。

(6) 中心点(CENTER)求线或者区域的几何中心点：NZS→PT。

5.2.3　空间对象近似

空间数据形状通常相当复杂，进行精确的数学表示极其困难甚至是不可能的，进而也难以在查询技术层面上进行空间关系的描述与实现。对于比较复杂的数据处理如数据查询，通常可以分为两步进行：一是排除掉大部分没有必要查询的数据；二是在剩余数据集上进行相应技术处理。例如，基于索引查询就是如此，即通过在数据集合上索引进行“近似”查询以实现查询目标的“必要”条件，再在剩余数据上进行“精确”查询以实现查询目标的“充分”条件。因此可以考虑空间对象的一种“近似”表示方法，即使用一种完全包围空间对象的简单几何图形如矩形、圆周或规则多边形来近似表达目标对象。这样，查询目标对象是否满足查询条件的充分必要条件问题就先转化为相应简单几何图形是否满足必要条件的问题。简单几何图形定位与运算相对简单，便于计算机实现，应用实践已经表明这样确实大大减少各种复杂的操作过程，提高系统效率。

1. 图形近似

空间对象近似分为“点”近似和“基本图形”近似两种情形。

1) 点近似

点是基本空间数据类型之一，非空间数据多属性查询也相当于多维空间点的搜索。某些规则图形也可用更高维空间点表示。因此，点近似是一种简单直观的降维方法。

(1) 一维空间线段$[a,b]$：使用二维空间点(a,b)表示。

(2) 二维空间的边平行于坐标轴矩形$\{(x_1,y_1),(x_2,y_2)\}$：使用四维空间点(x_1,y_1,x_2,y_2)表示，其中(x_1,y_1)、(x_2,y_2)分别为矩形的左下角和右上角坐标。

2) 基本图形近似

空间对象的近似基本图形主要有下述几种类型。

(1) 最小限定矩形：使用和坐标轴平行的“矩形”包围目标图形，这里矩形可以是高维情形。

(2) 最小限定圆：使用圆周包围目标图形，这里圆周也可以是高维情形。

(3) 最小限定多边形：使用规则多边性包围目标图形。

上述“最小”是指所有相应基本图形中的“最小”者，即再小就不能包围给定空间对象。

上述“最小限定矩形”方法是讨论空间对象常用的和基本的近似方法。

包含目标图形的边平行于坐标轴的最小矩形称为目标图形的最小限定矩形(Minimum Bounding Rectangle,MBR)。设二维 MBR 左下角坐标为(x_1,y_1),右上角为(x_2,y_2),则 x_1 和 y_1 就分别为空间对象的最小横坐标和纵坐标,x_2 和 y_2 分别为空间对象的最大横坐标和纵坐标。由上述“点近似”可知,MBR 可以使用其对角线上的两个顶点表示。因此,MBR 的意义实际上是将一般空间形体使用规则图形 MBR 近似表示,进而再将 MBR 使用两个顶点进行表示,从而是将空间形体使用了两个点进行近似表示。

MBR 不仅可以近似表示区域,还可以近似表示线段。更进一步,不仅单个空间对象可以用 MBR 近似表示,有时 MBR 还可以包含多个空间对象。最小限定矩形如图 5-6 所示。

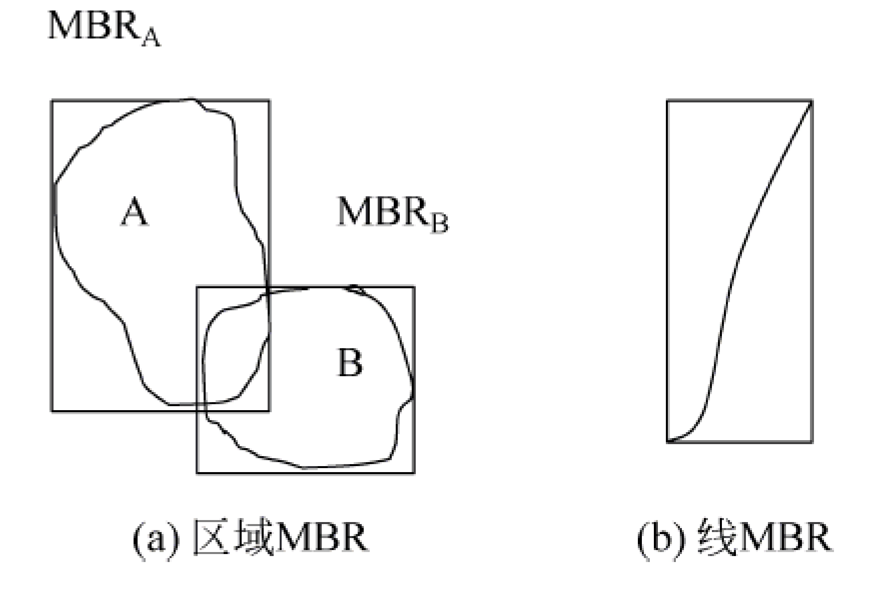


图 5-6　最小限定矩形

如果一个 MBR 还含有另外的 MBR,可称其为目录 MBR,否则就称为对象 MBR。

如前所述,使用 MBR 近似表示空间形体,能够有效实现对空间形体相互关系的必要条件判定。例如,如果两个空间对象相交,相应的 MBR 也相交;如果两个 MBR 不相交,则对应的两个空间对象也不相交。这样,用 MBR 代替空间对象检查相交情况,就可以排除一批不相交的对象,实现查询的必要条件。当然,两个 MBR 相交,并不能得出对应的空间对象一定相交,此时还需要用精确方法(高级程序语言编程的方法等)对 MBR 相交的空间对象逐个进行检验,找出真正相交的情形。先用高效率的近似方法进行筛选,再用精确方法进行确认以实现查询的充分条件,这实际上是包括 SDB 在内的高级数据库中常用的一种查询搜索方式。

基于基本图形近似的空间数据索引技术主要有 R-tree 及其各种拓展形式。

2. 栅格近似

用栅格(grid)近似表示空间对象,类似于用点阵和像素阵列等表示二维图像,原则上可以推广到高维空间,但主要用于二维空间。栅格近似可以有效使用在镶嵌模型当中。

基于栅格近似的空间索引技术主要有 G-file、R-file 和 Z-排序等。

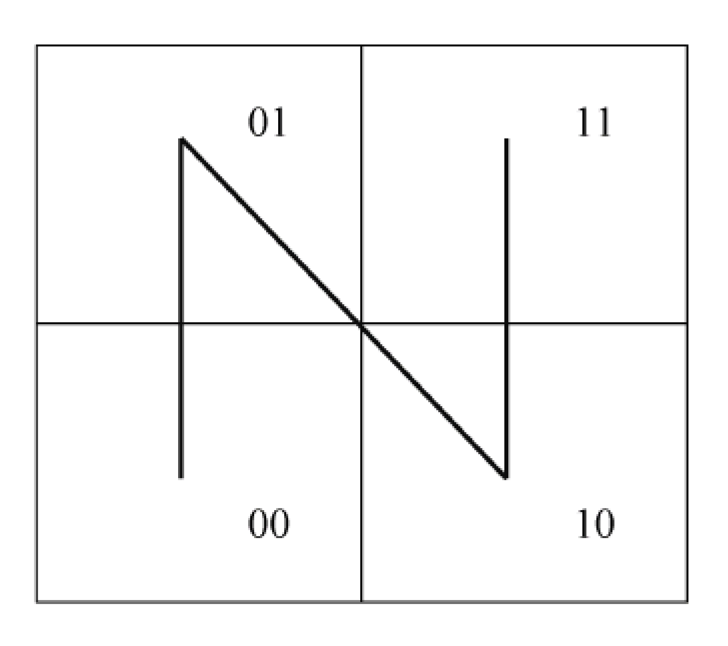
栅格近似技术要点是对其中所有像素基元的进行编码排序以有效地表示空间数据和描述空间对象相互之间的关系。

下面主要以图 5-7 所示的 Z-排序讨论像素基元的编码排序。

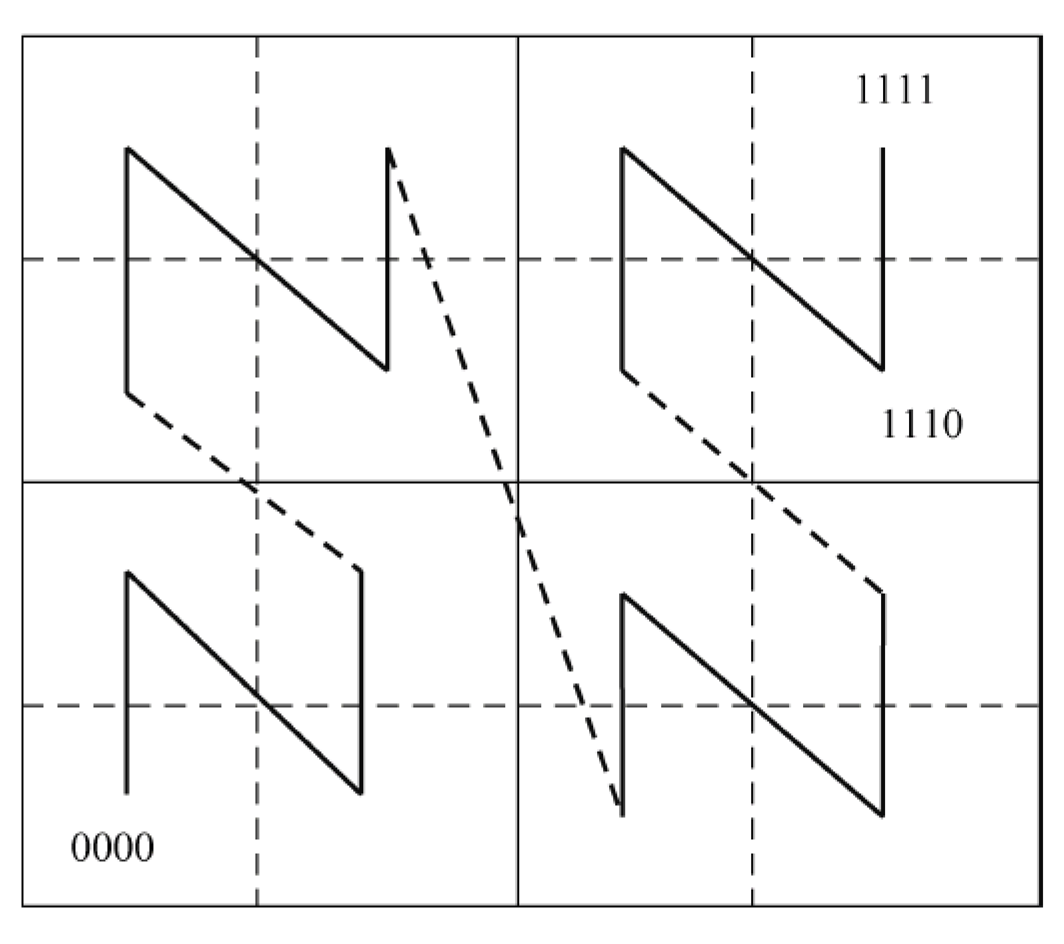
在图 5-7 中,用两位二进制串表示二维空间的 4 个象限,象限编码次序恰好是 Z 形。

每个象限还可以再分为 4 个象限。如此递归下去,直至方格分得足够细,整个空间看似一个栅格,可用来近似表示二维图形。这些方格用同样的规则编码。

在图 5-7(b)中,有个方格注有 1110,这就是该方格的编码。该方格位于图 5-7(a)的 11 象限中,所以编码前两位是 11。该象限又被一分为四,该方格位于再次剖分后的 10 象限,所以方格编码的后两位是 10。不难看出,图 5-7(b)中 16 个方格编码从 0000 开始,按图 5-7(b)中连线的次序,依次增至 1111。连线外形是一个 Z 形。按这种方法继续剖分和编码,栅格中每个方格按 Z 形连线排序。这种编码排序就是 Z 序(Z-order)栅格。



(a) 第1趟编码



(b) 第2趟编码

图 5-7　Z 序栅格

Z 序栅格中每个区域都可近似表示为二进制串的集合并称其为该区域 Z 元素(Z-element)。考察两区域相交或包含与否时,就可以按序比较两个区域的 Z 元素。

如果一个二进制串是另一个二进制串的前缀,则后者所代表的域包含于前者所代表区域,如 100101 一定包含在 1001 中。

下面是 A、B 区域 Z 元素,经比较可找出它们重叠部分。

A_1　　A_2　　A_3

A:0110,10, 10010,100110,10110,…

B:0111,100, 1010, 1011, 1101,…

B_1　　B_2　　B_3

重叠部分为(A_1,B_1)、(A_1,B_2)、(A_1,B_3)、(B_1,A_2)、(B_1,A_3)、…。

在上面的重叠栏中,二元组的第一项覆盖第二项。

用 Z 次序的栅格表示区域,将二维空间对象用一维有序二进制串序列表示,并且该字符串作为二进制数列还是单调增加,因此空间对象所在区域就可以以二进制串作为索引键,组成一棵 B-tree。

3. 基本图形空间关系

如前所述,空间数据管理的基础是通过基本图形近似来表示实际空间图形,此时,空间

对象相互间的空间关系就可以转换为基本图形的空间关系进行“先期”处理。基本近似图形中的线段和 MBR 最为常用，下面讨论基于线段和 MBR 之间相互空间的关系。

1）一维空间线段间关系

一维空间中两个线段具有如图 5-8 所示的 7 种可能关系，分别用记号“＝”“[”“%”“]”“/”“|”“<”表示。

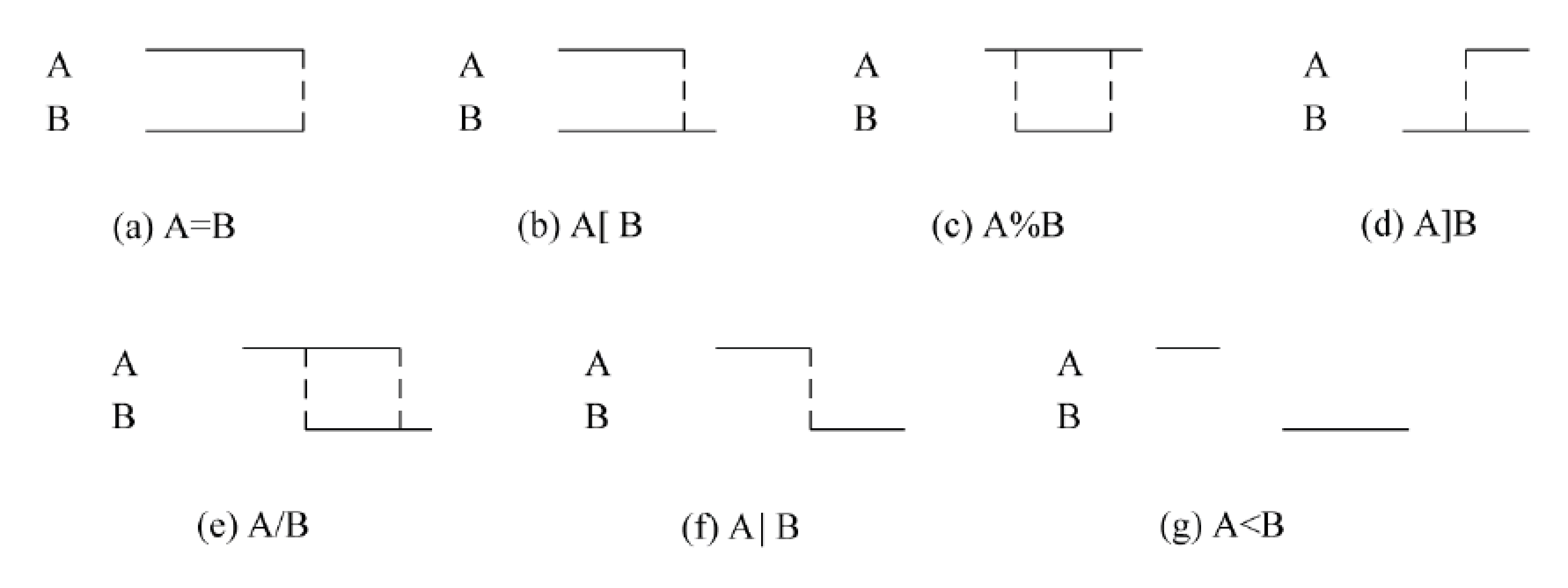


图 5-8 一维空间线段的关系

图 5-8 中(a)～(e)是相交关系，(f)和(g)是非相交关系。

设 A、B 线段的起点和终点分别为 x_{1A}、x_{2A}、x_{1B}、x_{2B}，则图 5-8 中(a)～(e)的关系可以归纳为 $\max\{x_{1A}, x_{1B}\} < \min\{x_{2A}, x_{2B}\}$。

2）二维空间矩形间关系

在二维欧氏空间中，一个矩形由其 4 个顶点的坐标确定。但是当一个矩形的两个相邻边分别平行于坐标轴时，只需要位于对角线上两个顶点即可确定该矩形。

矩形之间的相互关系可分为相交、相离和包含 3 种情形，其中包含可看作相交的特例。

设 A、B 为这种矩形，其左下角坐标和右上角坐标分别为 $\{(x_{1A}, y_{1A}), (x_{2A}, y_{2A})\}$ 和 $\{(x_{1B}, y_{1B}), (x_{2B}, y_{2B})\}$。则可以得到如下 A 和 B 空间关系的判定条件。

(1) 相离。如图 5-9 所示，当 A 和 B 在坐标轴上有一组投影不相交，则 A 和 B 就不会相交。

由此可得，A 和 B 相离的充要条件为：

$$y_{2B} < y_{1A}(\text{A 在 B 上方}) \vee y_{1A} < y_{1B}(\text{A 在 B 下方})$$
$$\vee\ x_{2A} < x_{1B}(\text{A 在 B 左方}) \vee x_{1B} < x_{2A}(\text{A 在 B 右方})$$

(2) 相交。如图 5-10 所示，如果 A 和 B 在 x 轴和 y 轴上的投影同时分别相交，则 A、B 相交。

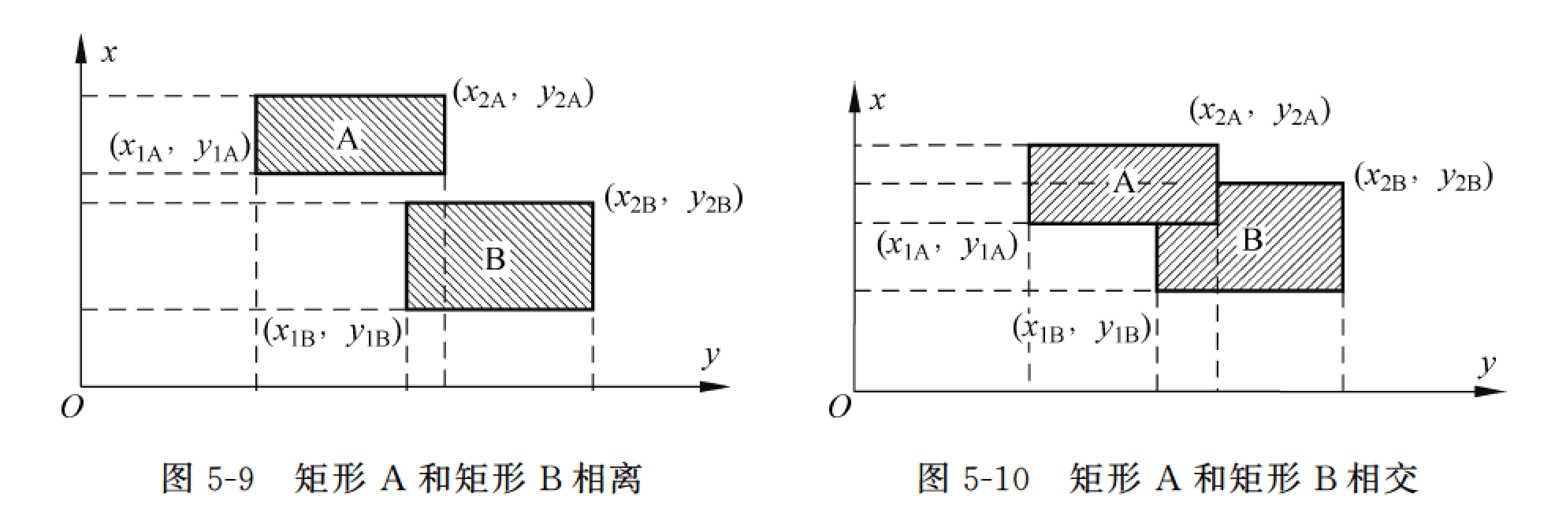


图 5-9 矩形 A 和矩形 B 相离　　图 5-10 矩形 A 和矩形 B 相交

由此可得，A 和 B 相交的充要条件为：

$$[\max\{x_{1A},x_{1B}\}<\min\{x_{2A},x_{2B}\}]\wedge[\max\{y_{1A},y_{1B}\}<\min\{y_{2A},y_{2B}\}]$$

(3) 包含。如图 5-11 所示，如果 A(B)在 x 轴和 y 轴上的投影同时分别包含 B(A)在 x 轴和 y 轴上的投影，则 A(B)包含 B(A)。

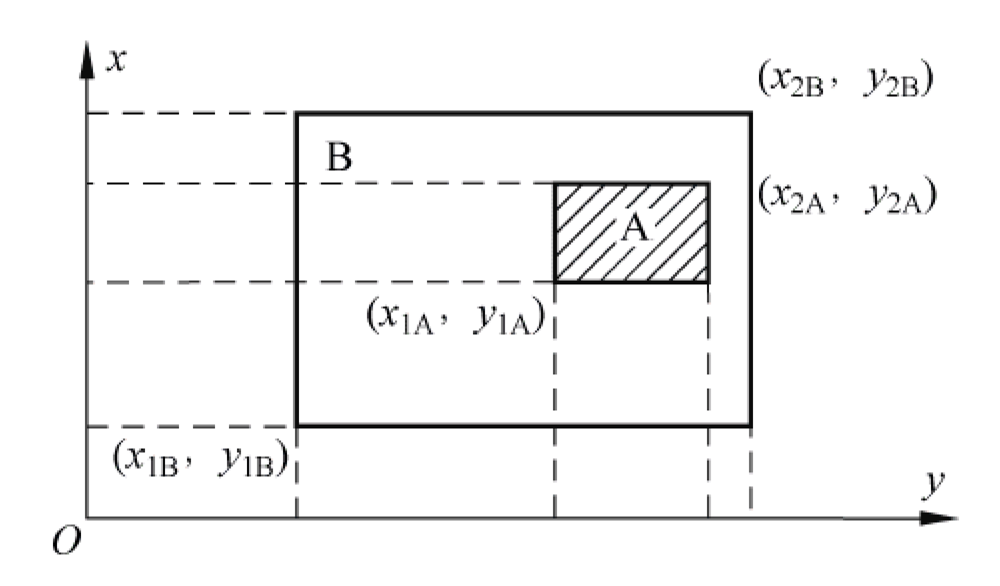


图 5-11　矩形 A 包含矩形 B

由此可得，B 包含 A 的充要条件为：

$$x_{1B}<x_{1A}\wedge x_{2A}<x_{2B}\wedge y_{1B}<y_{1A}\wedge y_{2A}<y_{2B}$$

5.3　空间数据库系统

RDB(Relation Database，关系数据库)主要适应于商业事务管理，难以有效地表示、存储、管理和查询具有多维特征的空间数据。例如，当需要查询“距离某超市 3km 范围内的顾客”，在 RDB 中，需要先将超市所在位置和顾客住址变换到一个便于比较和计算距离的参照坐标系中，然后系统扫描整个顾客列表信息，计算顾客住址到超市位置距离，通过谓词进行选择，当顾客住址到超市距离小于 3 公里，则输出顾客信息，否则继续搜索，直到所有顾客数据具备查询。此时，由于空间数据排序比较困难，传统数据库优势如查询优化等不能有效发挥作用，导致查询效率低下。当然可以使用编程技术设计相应的应用程序，但与传统数据管理情形类似，过多的应用程序将导致系统接口众多，使用复杂，安全性和数据一致性难以得到保障，同时也不符合一般数据库技术的基本要求。开发能够有效管理空间数据的专用数据库系统就显得十分必要，这就是空间数据库系统(Spatial DataBases System，SDBS)。

5.3.1　SDB 技术

SDB 技术通常包括下述主要内容。

(1) 空间数据模型：基于实际应用，通过适当归纳，引入各种必需的空间数据类型，并讨论相应的数据操作。

(2) 空间数据索引：空间对象之间难以定义“顺序”，所以空间数据索引就成为 SDB 技术的一个重要课题，在这方面已经取得了相当成熟的结果，并且应用到其他的领域。

(3) SDB 管理系统：空间数据模型和当前主流数据模型——关系数据模型具有较大的差异，需要研究如何在 RDBMS 基础上有效扩充空间数据管理功能的问题。

当然，SDB 无论是理论还是技术都还处于发展完善过程当中，而实际应用的需求又非常迫切，同时常规数据库(关系数据库)仍然是当今主流数据库，所以目前 SDB 多是作为传

统关系数据库的扩充方式出现和运作。

从本体论角度，研究和开发空间数据管理技术的意义主要包括下述几个方面。

(1) 空间是物质存在的基本方式。任何事物都有时间、空间形式和基本特性。作为反映现实世界的数据库，时间和空间是数据的基本属性。即使在某些应用中，时间或空间属性没有显式表现出来，但数据也"本质"地表示相应实体的当前状态和当前位置。因此，从根本上讲，所有数据都应当既有非空间内容又有空间内容。例如，一个地区或国家至少有一个非空间数据，如国家名和一个空间数据，国家的边界。

(2) 空间数据是实际应用的基本形式。在某些重要的应用当中，研究对象的空间关系成为查询或处理的主要内容，如天文、地理信息系统 GIS、城市规划、管道及网络系统、交通图、大规模集成电路版面设计、分子结构图和医学图片等。面向这类应用的数据库系统必须在常规数据库系统的基础上，增加相应空间数据类型及其相关操作，提供空间素材及面向空间应用的交互式图形用户界面。

(3) 非空间数据可作为空间数据处理。在许多实际研究和开发过程中，数据尽管不属于空间数据类型，但也可作为多维空间问题处理。例如，在关系数据库中，随着普遍使用多属性查询，如果将每个被查询的属性看作查询空间的一个维数，则多属性查询便可归结为多维空间的搜索；在时态数据库中，可以将相互正交的"事务时间"和"有效时间"看作二维的空间数据，进而应用 SDB 技术(如空间索引技术)进行相关的基本讨论。

5.3.2 SDB 结构

早期的 SDB 处理系统(如 GIS)一般建立在文件系统基础之上，所有空间数据定义和处理都由应用软件解决。目前仍有这种系统在运行，但是新开发的系统多以 DBMS 为基础，最初以 RDBMS 为主，后来以 ODBMS 为主。下面简述这两种 SDB 系统结构。

1. 基于 RDBMS 系统结构

由于目前 RDBMS 产品还不能充分支持空间数据类型及其操作，解决这个问题一般有如下两条途径。

(1) 分层机构。这种系统结构中的方法一是在所支持的数据类型"上方"增加一层以实现空间数据类型及其操作。用所支持的数据类型表示空间数据能够方便地借用关系数据模型表示，但其缺点是构造复杂空间数据的开销很大、性能不好。方法二是以二进制数据类型即图片方式表示空间数据存放在 RDB 中，如存放一张地图或遥感图片。这虽然可以避免用点、直线段等简单数据类型表示所有空间数据，但字段只能存放原始数据，对数据的解释还是得依靠上层解决。

(2) 双重结构。分层结构只能利用的已有功能对空间数据做出有限支持，而 SDB 存储结构和索引等不可能由上层来实现。为了克服此缺点，可以增加一个专门管理空间数据的子系统。有关空间的数据可分为两个部分：非空间属性，存于 RDBMS 中；空间属性，存于空间数据子系统中，如图 5-12 所示。两者之间用逻辑指针(即空间属性标识符)相连。这样用户看到的却是一个不可剖分的元组。图 5-13 表示一个地区 A 的元组。前两个属性地区名和人口是普通属性，第三个属性是该地区的地图，属于空间属性。上层接口在插入元组时，先将第三属性用该地区地图标识符代替，且将此元组存于数据库当中。该地区地图存放于 SDB 子系统中。在查询时，首先从关系数据库中查出表示该地区的元组；然后根据第三

属性标识符，在空间数据子系统中检索该地区地图，且由上层接口装配成表示该地区的元组。双重结构比起分层结构有所前进，但由于多出了一个分解和合成的步骤，增加了开销，更主要的是缺少了全局优化。例如，在查询时，有时既可以采用空间属性索引，也可以采用非空间属性索引，此时，上层接口无法对此做出选择，因为难以确定二者的执行代价。

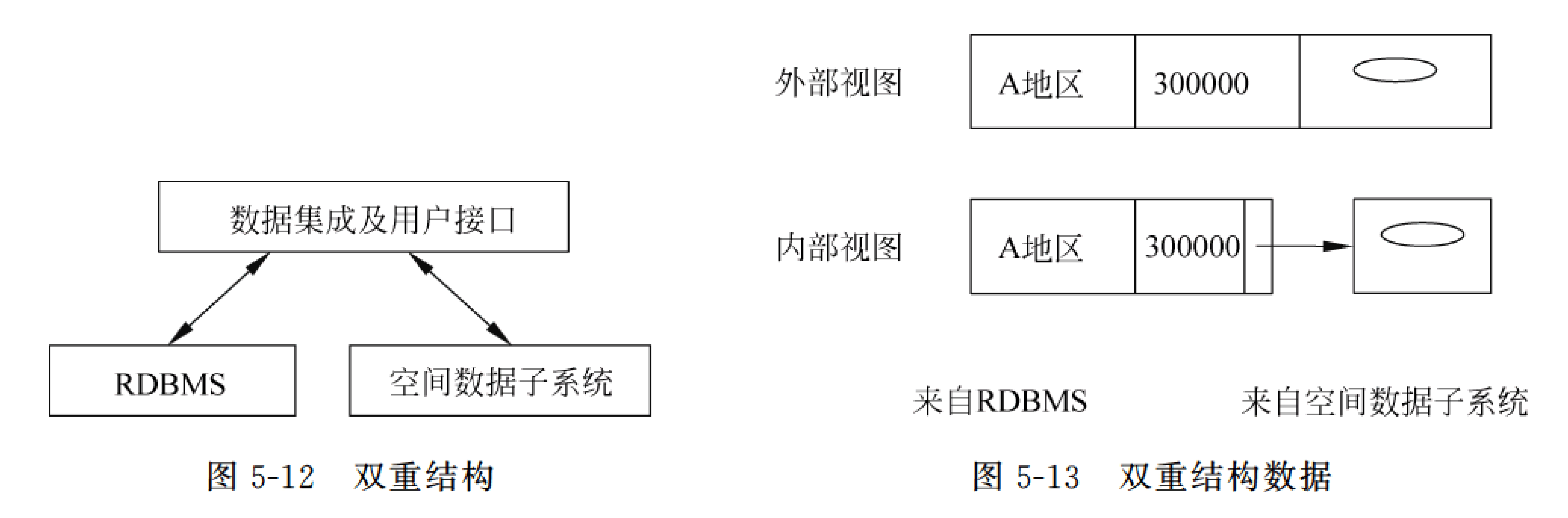


图 5-12　双重结构

图 5-13　双重结构数据

2. 基于 OODBMS 或 ORDBMS 系统

由于面向对象数据库管理系统已有部分商业化产品，而且其中多可以进行扩充，因此能够在其中定义空间数据类型和操作。空间数据和非空间数据可以用相同方式进行处理，这就为研制集成 SDB 系统创造了基本的技术条件。

但 OODBMS 或 ORDBMS 仅仅是提供了扩充成空间的可能，实现此目的还需做大量的工作。例如，定义空间数据类型及其操作，实现空间索引及有关空间数据的用户接口，增补有关空间数据的查询优化策略等。从发展来看，建立在 OODBMS 或 ORDBMS 之上的 SDB 系统将会逐步成为技术发展的主流。

5.4　空间数据查询

空间数据操作的特色是查询的复杂性和多样性，其中有些查询可以借鉴或通过关系数据操作实现，但从整体上考虑，需要研究和建立不同于关系数据查询的框架与技术。

5.4.1　空间数据查询操作

空间数据查询是指在 SDB 中查找满足一定条件的空间数据对象的操作过程。

由于空间数据远比常规数据复杂，当前并没有成熟的空间数据代数运算系统，而各种查询算子也依赖于应用背景，因此当前空间数据查询操作大多采用如下两种实现途径。

(1) 基于关系数据查询。允许使用抽象数据类型表达空间数据对象和相应的基本运算，而且查询结果通常也同关系数据库情形一样为满足一定查询条件的空间数据关系表。这种查询方式得以实现的前提是能够实现空间数据的关系存储。

(2) 基于索引查询。主要是基于 R-tree 的查询，此时，空间数据基于空间数据模型自身的形式而无须转换为关系表。它能够有效完成各种基于拓扑关系的查询从而成为 SDB 最重要和最基本的空间数据查询技术，同时还可以用于时空查询和时态查询。

1. 空间查询类型

空间数据的基本类型要素是空间点和空间区域，相应的查询类型也照此划分。

(1) 空间点查询：给定一个确定的空间点对象，查询所有包含该点的空间数据对象。

(2) 空间区域查询：给定一个确定查询区域，查询所有和该区域至少有一个公共点的空间数据对象。

空间区域查询情形比较复杂，通常可以分为下述情形。

(1) 相交查询：给定一个确定查询区域，查询所有和该区域相交的空间数据对象。

(2) 包含(被包含)查询：给定一个确定查询区域，查询所有包含该区域(被该区域包含)的空间数据对象。

(3) 相邻查询：如果两个区域只有部分或全部相同的边界而没有其他相交部分，则称这两个区域相邻。相邻查询就是给定一个确定查询区域，查询所有与该区域相邻的空间数据对象。

(4) 范围查询：给定一个确定查询区域，查询所有与该区域距离小于给定值空间数据对象。

(5) 最近邻居查询：给定查询空间对象，查询和其距离最小的所有空间数据对象。

不难看出，上述各类查询实际上都涉及两个空间对象相交的情形，因此，相交查询是空间数据查询操作的基础。

2. 基于关系数据的查询实现

空间数据查询的实现可以有不同的技术实现方式。如果采用 RDB 扩充方式管理空间数据，则可以由关系数据查询方式实现相应的空间数据查询。例如，空间关系的界定在于相应选择条件或连接条件中出现空间谓词，这与关系代数中相应情形实际上并没有根本上的不同。以下对比关系运算中相关操作介绍空间选择和空间连接。

1) 基于关系代数查询

【例 5-1】 通过空间选择完成下述查询。

(1) 选择广东省所有城市。

$$\sigma_F(\text{城市})$$

其中，F＝CENTER(城市地图)INSIDE 广东。

城市是关系名，其中有属性“城市名”“人口”“城市地图”。城市地图表示市区及其周边地区，“广东”是一个区域名称。CENTER 是 SDB 中的一个函数，用来计算城市地区的中心点。显然，如果城市中心点在广东省区域内，则该城市一定属于广东省。

(2) 选择流经广东省所有河流。

$$\sigma_F(\text{河流})$$

其中 F＝ROUTE(河流)INTERSECTION 广东。

“河流”是关系名，其中有属性“河流流域图”。ROUTE 是 SDB 中的一个函数，计算河流、道路等的中心线。

(3) 选择距离广州小于等于 100 000 米和人口大于等于 50 万的所有城市。

$$\sigma_F(\text{城市，广东区域图})$$

其中 F＝DIST(城市名，广州)<＝100 000 AND 人口>＝500 000。

城市是个关系，“广州”是城市名，F 中的第一个谓词是空间谓词，要用到广东省地图。

【例 5-2】 通过空间连接对每条河流找出沿河 10 000 米的所有城市。

设“河流”“城市”是两个关系。在关系“河流”中，有属性“河流流域图”。如果城市中心

距离河流小于等于 10 000 米，则该城市和河流匹配。可以用空间连接表示如下：

$$\Pi_{\text{河流名,城市名}}(\text{河流} \bowtie_F \text{城市})$$

其中，F=Mindist(城市名，ROUTE(河流流域图))< 10 000。

在进行连接时，在“河流”的属性“河流流域图”中，检查各个城市是否满足连接条件 F，如果满足，则两者匹配。

2）基于 SQL 查询

在 SQL 语言基础上可以扩充空间数据类型及其操作和相应关键词，但由于这些扩充与应用有关，目前还未形成标准，以下列举一些例子予以说明。

【例 5-3】 基于 SQL 完成下列查询。

(1) 选择广东省所有城市及其人口。

```
SELECT 城市名,人口
FROM 城市
WHERE CENTER(城市地图)INSIDE 广东省
```

(2) 选择流经广东省所有河流的河流名及其在广东省境内的长度：

```
SELECT 河流名,LENGTH(INTERSECTION(ROUTE(河流流域图),广东))
FROM 河流
WHERE ROUTE(河流流域图)INTERSECTS 广东
```

(3) 选择距离广州小于等于 100 000 米，人口大于等于 50 万的所有城市：

```
SELECT 城市名,人口
FROM 城市,广东区域图
WHERE DIST(城市名,广州)<= 100 000 AND 人口 => 500 000
```

【例 5-4】 将例 5-2 中查询用 SQL 风格表示。

```
SELECT 河流名,城市名
FROM 河流,城市
WHERE MINDIST(城市名,ROUTE(河流流域图))<= 10 000
```

SDB 查询的开销一般比关系数据库大，特别是空间谓词求值的开销远比数值或字符串的比较要大。若采用顺序扫描方法进行查询，则效率就会很低，因此采取更为有效的空间索引是十分必要的。

5.4.2 空间数据索引

传统关系数据索引技术多是针对一维属性数据提出的，基本思想是基于关系表主键进行相应设计，难以直接用于空间数据索引。实际上，在过去的几十年期间，空间数据索获得数据库业界内众多学者关注，相继提出了基于空间目标位置信息的索引结构与查询算法。其中以 R-tree 为代表的各类空间数据索引技术更成为空间数据管理的重要亮点和基本特色。图 5-14 简要表示了空间数据索引技术的基本类型与发展。

空间数据具有多维特征，相应空间数据索引是多维数据索引，这有着与常规数据索引不同的基本特征。

1. 空间索引特征

从计算机处理的角度来看，空间数据具有下述两个特点。

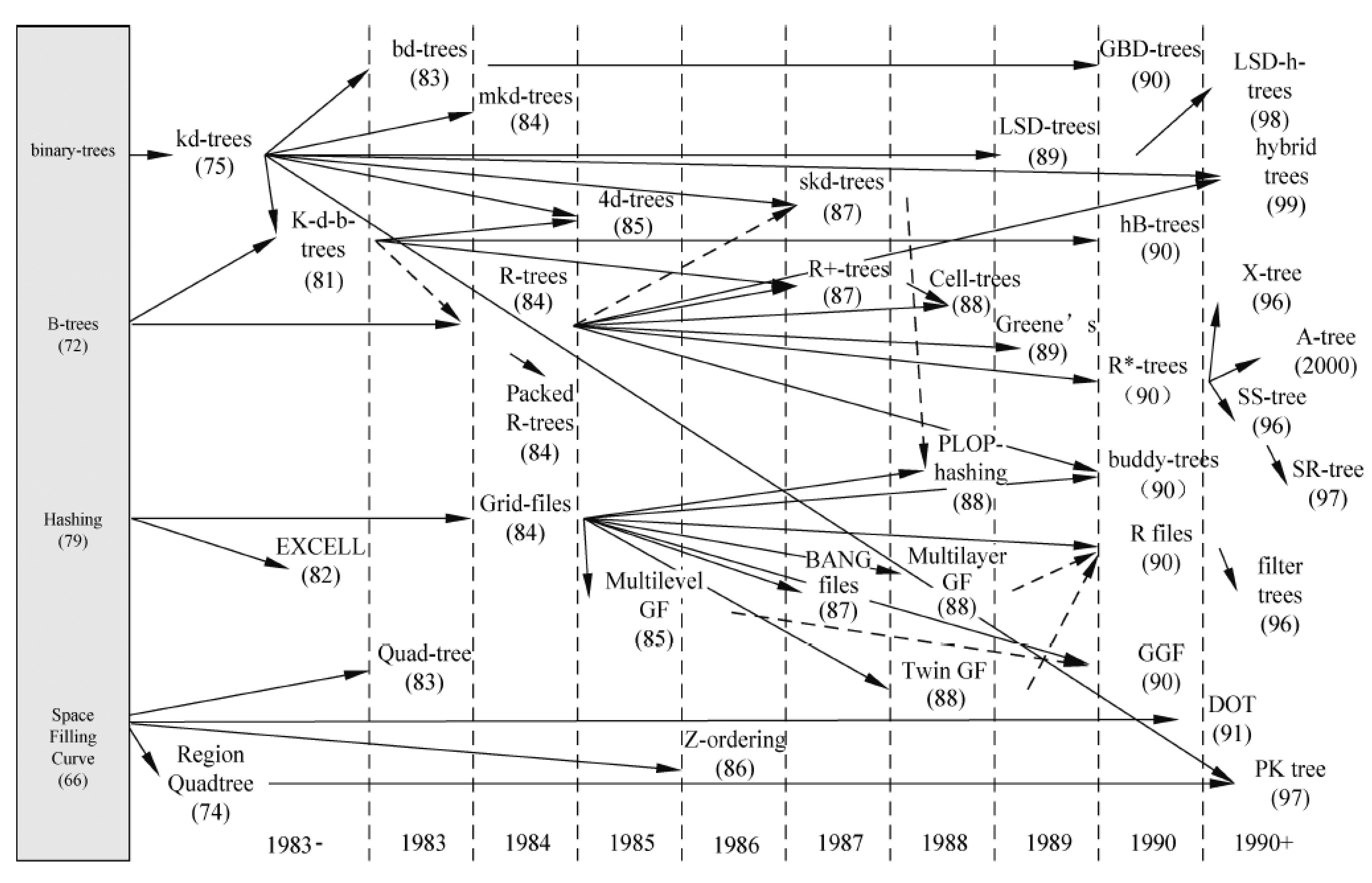


图 5-14 空间数据索引技术的基本类型与发展

（1）空间数据形状复杂。直接进行表示和实现操作需要多方面考虑和复杂算法设计，因而属于大强度“CPU 密集型”。

（2）空间数据体量庞大。实际操作过程中通常需要调用大量数据进入内存，因而又属于高浓度的“I/O 密集型”。

正因为如此，如果在查询过程中直接进行精确位置判断和相应的关系查询运算（如判断两个空间数据对象是否相交），可能需要量消耗大计算机资源，而且即便如此也不一定能够实现相关操作。但我们知道，通过在数据集合上建立索引进行数据查询，实际上就是寻找目标数据应当满足的必要条件。而空间索引结构大多采用如前所述的空间目标的近似方法，即使用一个完全包围目标的简单几何图形，如矩形、圆周或规则多边形来近似表达目标对象。这样，目标对象满足查询的必要条件就转化为相应简单几何图形满足必要条件的问题。简单几何图形的定位与运算通常都相对容易。因此，可以认为空间形体的近似表示实际上就是为建立空间数据索引提供了技术支撑。按照上述考虑，实际中进行空间数据查询可以分为两个步骤进行。

1）近似图形过滤

通常都是针对空间数据的近似图形（如 MBR 数据或栅格数据）建立索引，同时将所需查询信息也转化为相应近似图形，通过查询信息近似图形与索引中数据近似图形之间关系取得过滤后的近似图形集合，此时排除了不满足查询“必要条件”的数据。显然，能够排除掉的数据集合大小与否是衡量相应空间数据索引效率的基本指标，经过过滤后的数据集合就成为所需查找数据集合的更为精准和范围更小的超集。此时，实际上是充分发挥数据库的“I/O 密集型”特征与优势。

2）准确数据筛选

通过基于实际应用场景的各种方法手段包括非数据库查询途径，如高级程序设计语言

编程方法在过滤后集合中进行精准查找，这通常需要建立相应的 CPU 密集型算法，因而是一个相当复杂并涉及相关领域知识的过程。通常认为是在 SDB 研究范畴之外，也就是说，空间数据索引主要是讨论近似图形的过滤部分。当然，对于某些情况（如图形数据存储在关系数据库当中）也可对过滤后数据集进行关系数据筛选。

由上述分析可知，空间数据索引操作通常得到的是所需查询结果集的超集，实际上这也是解决类似复杂问题的基本思路，因为如果直接处理空间数据，不要说相应查询算法极其复杂，就是给出相应的数学表示都难以做到，更不用说给出能够进行机器处理的计算机描述，直接处理空间数据对于数据库而言应该是难以企及的。尽管通过空间数据索引得到的还不是准确结果，但其数据规模会大大缩小，能够有效缩小查询处理的"I/O 密集型"的强度，进而为"CPU 密集型"开辟了技术实现的有效途径。

现有空间索引一般选取图形点和图形 MBR 作为索引对象。与通常索引相比，空间索引具有如下特点。

（1）索引对象无序性。空间索引对象一般难以定义明确索引顺序，当确认了某个对象在一个子空间内时，若要进一步搜索，通常需要对该子空间内的数据逐个进行遍历比较

（2）索引对象交叉性。空间索引对象可以交叉，甚至能够重叠，一个索引对象可以属于多个子空间即索引结构中的多个不同分支。也就是说，一个空间数据结点可以同时属于多个目录索引结点，因此查询进程更为复杂，更新过程也更为精细。在查询操作时，通常需要进行多路搜索并对建立后的索引进行必要的优化处理，在更新过程中需要考虑多重因素。

2. 空间索引分类

空间索引通常有两种分类方法：一种是基于现有技术的实现途径，此时空间索引可分为基于树的索引和基于 Hash 方法的索引两种类型；另一种是基于索引数据对象自身特征，将空间索引分为基于空间点对象和基于空间非点对象两种情形。

1）基于树和 Hash 方法索引

（1）基于树的索引主要包括有基于二叉树、基于 B-Tree 和基于四叉树 3 种情形。

① 基于二叉树的空间数据索引：如 BD-tree 和 Kd-tree 索引等。

② 基于 B-tree 的空间数据索引：如 R-tree 索引技术等。

③ 基于位置链的四叉树空间数据索引：如 Filter-tree、Z-排序等。

（2）基于 Hash 方法的索引主要包括 Grid-文件等。

2）基于空间点和空间非点对象索引

空间点对象和空间非点对象的区别在于：前者只有位置而没有自身度量，而后者既有位置又有自身度量。当然，在某种意义下，点也可看作空间图形特例，因此，非点空间图形索引技术也适合于点图形。但一般而言，点对象和一般空间对象在技术实现上有较大不同，需要分别讨论。

（1）空间点对象索引。

多维空间点索引可分为基于二叉树和基于栅格两种情形。

① 基于二叉树的空间点索引：基于二叉树索引结构的点查询空间索引主要有 Kd-tree、KdB-tree 和 LSD-tree 等，而 Kd-tree 可以看作是其中的主要代表。为将 Kd-tree 存储组织到外部存储器，需将 Kd-tree 与 B-tree 结合，由此就产生了 KdB-tree 方法。

② 基于栅格的空间点索引：基本思想是将索引空间划分为相等或者不相等的一些小型栅格，与每个栅格相关的空间目标存储在同一磁盘页内，栅格访问地址可以直接通过求数

组下标或某种适当算法得到。这种方法主要代表是 G-file 和 R-file 技术。

(2) 空间区域对象索引。

空间区域对象索引多采用基于 B^+-tree 技术。

① 基于 B^+-tree 拓展：由于 B^+-tree 广泛应用于数据库系统并且具有优秀表现，而现有 SDB 在技术实现方面大多依赖于常规数据库系统，因此主要空间数据索引技术大多基于 B^+-tree 基本思想。例如，著名的 R-tree 就是典型代表。与 B^+-tree 不同，R-tree 中的索引项有可能存在前述的数据重叠。为了提高系统效率，需要研究如何达到某种意义下的“最小”重叠，因此又产生了 R-tree 的拓展和变种 R^+-tree 和 R^*-tree 等

② 转化为 B^+-tree 处理：这些技术的基本点也是将索引空间进行剖分，对所有剖分后的小区域进行适当编码，通过编码将剖分区域进行排序，从而完成高维区域到有序编码集合的一一映射。有了编码集合，就可以应用 B^+-tree 方法进行编码的索引，从而完成对高维对象的索引工作。这里关键之处在于剖分区域集合到编码集合的映射。常用映射有 Z-排序、Hilbert 曲线和位置键方法。

下面主要按照空间对象自身几何特征讨论相应的数据索引技术。

5.5 空间点索引技术

Kd-tree(K-dimension tree)是由 Bentley 于 1975 年提出的一种 $k \geqslant 2$ 维的二叉查询树(BST)技术。与常规二叉树索引不同，Kd-tree 中第 j 层中的每个结点都看作是 k 维空间中第 $j \bmod (k)$ 维坐标空间中一个点。这里，将 $j \bmod (k)$ 看作是所谓分辨器函数 $d(x)$ 在层数 j 处的函数取值 $d(j), j=0,1,2,\cdots,m$，其中 m 是给定二叉查询树的层数。在构造和查询过程中汇总，二叉树的每一层都根据该层的分辨器做出分枝决策。

5.5.1 Kd-tree 和 KdB-tree

基于二叉树索引技术主要代表有 Kd-tree 和 KdB-tree。

1. Kd-tree

Kd-tree 是一种基于二叉查询树(BST)处理 $k(\geqslant 2)$ 维空间点数据的索引技术，其特点是树中每个结点都表示 k 维空间中的一个点，并且规定树中根结点位于树的第 0 层，根结点的子结点位于树的第 1 层，以此类推。

Kd-tree 的基本概念是分辨器(discriminator)函数。

分辨器函数：设需要处理 $k(\geqslant 2)$ 维空间的数据对象，j 表示 Kd-tree 中的第 j 层，Kd-tree 分辨器函数定义为：$d(j)=j \bmod (k)$。

由定义可知，分辨器实际上就是树中层数 j 基于空间维数 k 的取模函数，函数值是 j 关于 k 的余数，而该余数 $d(j)$ 在 0、1、2、…、$k-1$ 中取值，通常 $d(j)=0$ 表示 k 维空间的 x_1 轴，$d(j)=1$ 表示 k 维空间的 x_2 轴，…，$d(j)=k-1$ 表示 k 维空间的 x_k 轴的语义。例如，对于 $k=2$ 和层数 j，$d(j)=0$ 表示二维空间中的 x_1 轴即 x 轴，$d(j)=1$ 表示二维空间中的 x_2 轴即 y 轴。

按照下述 Kd-tree 构建过程，分辨器函数的作用在于确定 k 维区域的下一趟剖分平面的选取，即 Kd-tree 中每一层依赖于其父结点层(第 j 层)分辨器做出相应的决策分枝。

1) Kd-tree 构建

Kd-tree 或者是一棵空树，或者是一棵由第 $j(\geqslant 0)$ 层开始满足下述条件的二叉树。

(1) 当第 j 层结点 n 的左子树非空时，该子树中所有结点的第 $d(j)$ 维坐标都小于结点 n 的第 $d(j)$ 维坐标值。

(2) 当第 j 层结点 n 的右子树非空时，该子树中所有结点第 $d(j)$ 维坐标都大于或等于其父结点的第 $d(j)$ 维坐标值。

(3) 相应结点的左右子树也是 Kd-tree。

从数学上来讲，上述 Kd-tree 的构造条件相当于对 k 维空间中区域 E_k 不断通过由分辨器函数选定的平行于某个坐标面 $x_{d(j)}=x_{d(j)0}$ 的 $k-1$ 维平面对 E_k 进行逐次剖分。剖分的终止条件是实现根据需要设定一个正整数阈值 α，当所剖分得到的子区域中数据点个数小于或等于 α 时就停止剖分。

例如，当对于二维空间数据而言，即当 $k=2$ 时，由于 $d(j)=0$ 表示 x 轴，$d(j)=1$ 表示 y 轴，则此时就是当 $d(j)=0$ 时，用直线 $x=x_0$；当 $d(j)=1$ 时，用直线 $y=y_0$ 分别对二维数据区域进行交替剖分。

【例 5-5】 设有如图 5-15 所示的空间数据点，设 $k=2$，Kd-tree 构建如图 5-15 所示。

(1) 由 $j=0$ 层的根结点 A 开始，其 x 坐标为 x_1：此时，$d(0)=0 \bmod (2)=0$，因此取 $x=x_1$ 进行剖分，则剖分线 $x=x_1$ 左边数据点(B、D、G)都是根结点(x_1)左子树结点，右边(C、E、H)是根结点(x_1)右子树结点。

(2) 对于根结点 A 子结点而言：$d(1)=1 \bmod (2)=1$，因此在根结点(x_1)左子树中选取 B，其 y 坐标为 y_1，在根结点 A 右子树中选取 C，其 y 坐标为 y_2 进行分别使用 y_1 和 y_2 对左子树和右子树进行剖分。

(3) 再重复上述过程，按照分辨器读数分别相应进行适当剖分处理，最终完成 Kd-tree 的构建。

由上述剖分得到相应的 Kd-tree 索引构造，如图 5-16 所示。

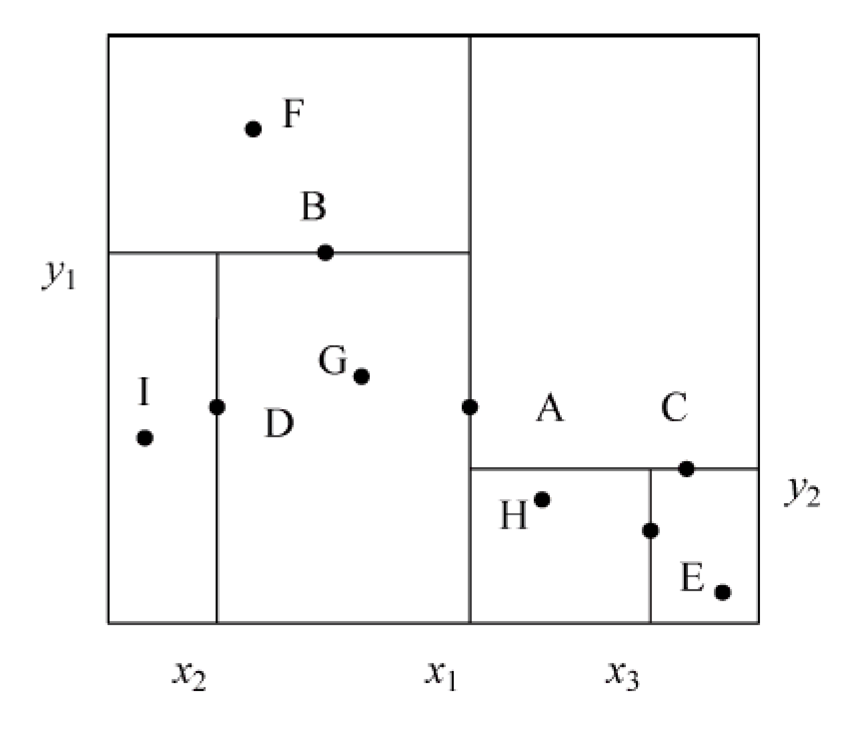


图 5-15　二维数据区域剖分

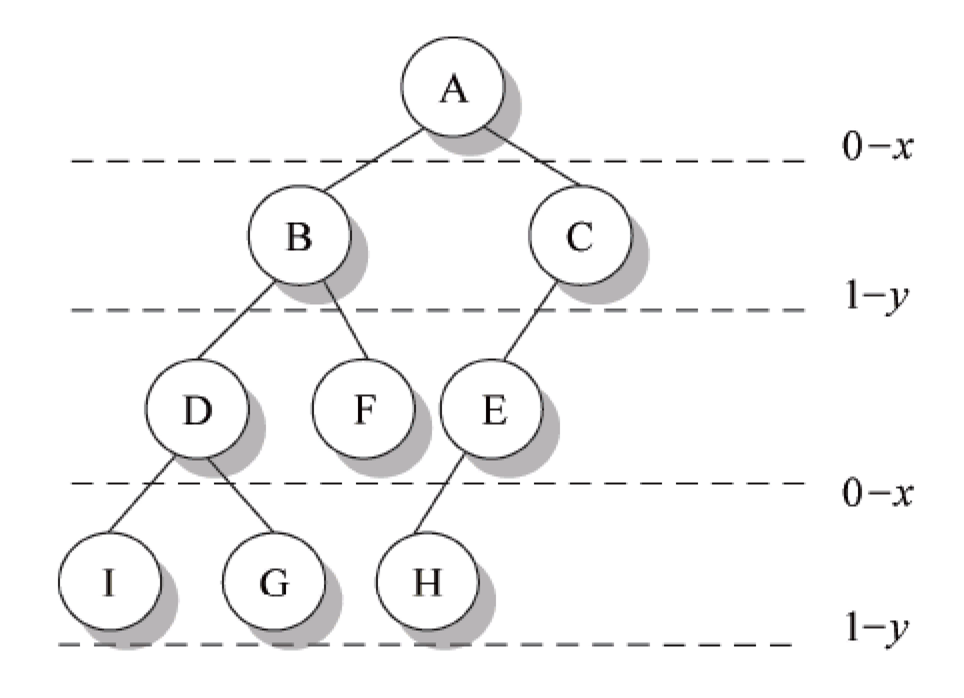


图 5-16　Kd-tree 索引构造

2) Kd-tree 数据操作

Kd-tree 的数据操作类似于二叉查找树的相应操作。

(1) 数据查询。设需要查询的空间点为(x_0,y_0)。由根结点 x_1 开始。如果 $x_0>x_1$，则向 x_1 的右子树搜索，否则向 x_1 的左子树搜索。不妨设 $x_0>x_1$，搜索到 x_1 的右子树，此时，如果 $y_0>y_2$，则向 y_2 的右子树搜索，否则向 y_2 的左子树搜索；不妨设 $y_0>y_2$，如此一直搜索到叶结点 D。如果 D 中存在(x_0,y_0)，则返回结果，否则表示不存在查询结果。

（2）数据插入。在插入结点时，可能引起结点分叉，如在 D 中插入一个空间点（x_3，y_3），则 D 中就有 4 个点，需将 D 再分为两个区域。例如，用 $x=x_3$ 作为剖分线，将 D 分为 D_1 和 D_2 两个部分，此时在相应索引中以虚线表示的子树替代 D 结点。

（3）数据删除。在删除结点时，可以允许存在空的区域，因此不需要修改数据区域的剖分和相应索引结构。

Kd-tree 具有下述不足。

（1）由于采用二叉树结构，索引树的层次较多。

（2）由于索引形状与插入次序相关，树结构可能不够平衡。

（3）由于索引没有按照页块组织，不适合在外存中建立结构。

针对上述问题，Rodinson 在 1981 年提出基于通过 Kd-tree 和 B-tree 结合而成的 KdB-tree 按照页块组织索引，解决了索引树的平衡问题；Hentich 等人也于 1989 年提出了 LSD-tree，改变了按照位数交替剖分数据空间的限制，也可保持索引树的平衡。

2. KdB-tree

B-tree（Balanced multi-way tree，平衡多分树）和 B^+-tree 的基本差异之一就是前者索引结点也是数据结点；而后者的索引结点只具有导航意义，数据都存储在叶结点当中。这种将索引结点和数据结点分开处理的方法可以有效提升索引的数据操作性能。正是基于这样的考虑，借鉴 B^+-tree 思想，就可以类似于 B-tree 到 B^+-tree 的拓展那样，将 Kd-tree 推广到 KdB-tree。

KdB-tree 的基本点是非叶结点（根结点和内部结点）都为索引结点并称为区域页（Region Pages），而叶结点为数据结点并称为点页（Point Pages）。也就是说，点页存储点数据目标，区域页存储索引子空间描述及指向下层页的指针。

KdB-tree 的一个实例如图 5-17 所示。其中，剖分过程与 Kd-tree 相同，只是剖分坐标选取不同，如为方便记，可以采用相应的等分法。在图 5-17(a)中，首先选取适当 x 坐标将所给数据结点剖分为是 s1 和 s2 两个区域页；其次对 s1 和 s2 分别选取适当 y 坐标剖分为 s11、s12 和 s21、s22 等的子区域页；再次对 s12 剖分为 s121 和 s122 等的子区域页；最后就得到如图 5-17(b)所示的索引树示意图。

由此可知，KdB-tree 借鉴了 B^+-tree 的特点，数据扇出大，树高比较小，对均匀分布数据效果好。不足在于：一块空间中无论有无数据都会被索引；频繁向下分裂导致空间利用率低；对非均匀分布数据效果较差。

KdB-tree 主要用于点查询，但也可用于区域查询。

（1）点匹配查询：需要遍历树的某一个分支，即访问所有索引子空间包含该查找点的区域页直到某个点页，最后提取点页中的点加以判断。

（2）区域查询：需要访问所有索引空间与查找区域相交的区域页即点页，查找路径往往是多条。

在 KdB-tree 中实施插入结点时，需要首先查找该结点应当插入的点页，如果该页未满，则简单插入该点；如果该页已满，则需要分裂该页，即将该点页一分为二，并使这两个点页包含同样多的点。当点页分裂时，需要在上级区域页中增加一项。由此可知，点页分裂可能导致父区域页分裂。同理，区域页分裂也可向上传播直至根结点。所以，KdB-tree 是一种平衡树。

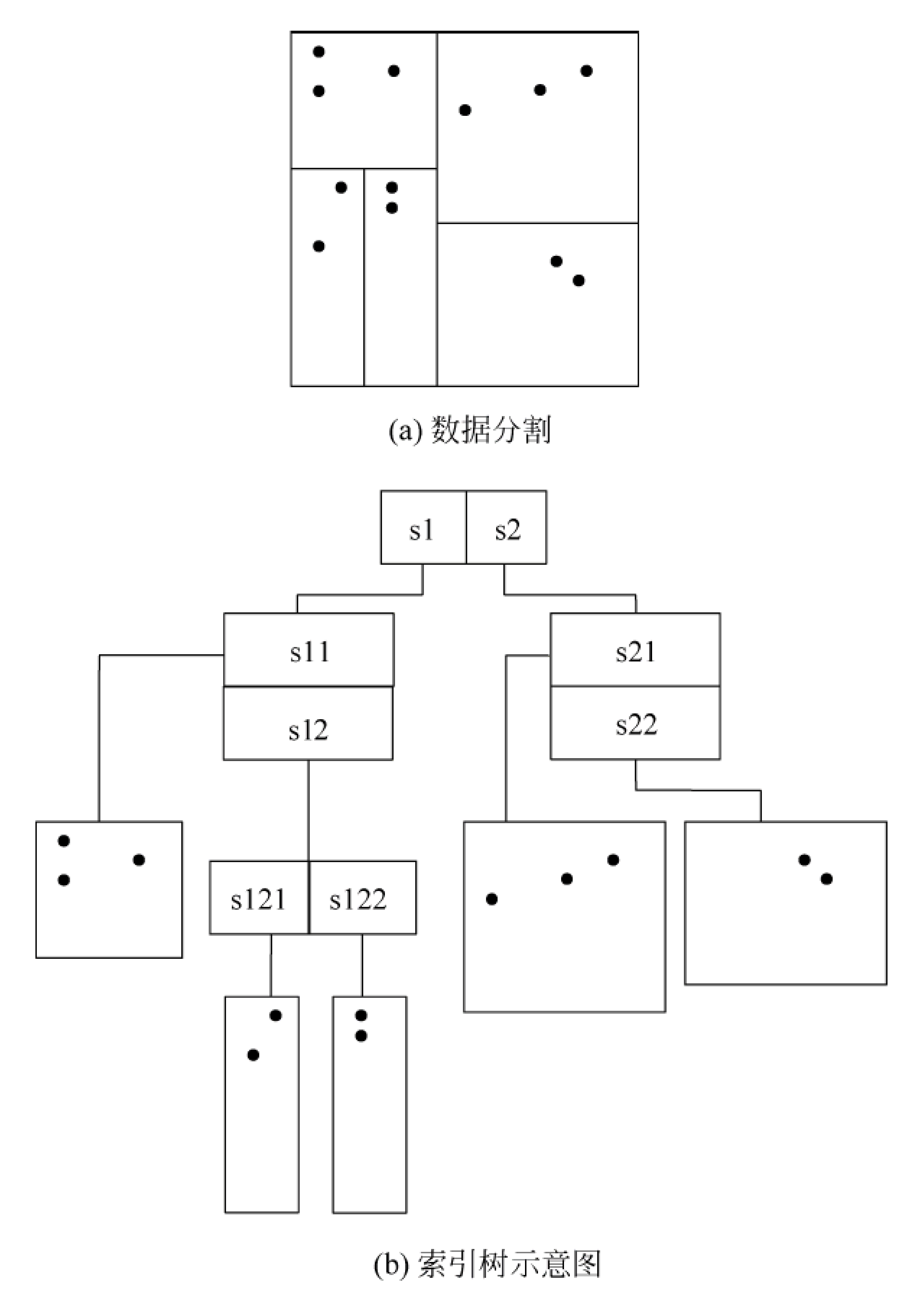


图 5-17　KdB-tree 的一个实例

5.5.2　G-tree 索引

G-tree(grid tree)是一种基于 B-tree 的动态索引技术,其基本要求是可动态地随数据变化而进行相应消长,并在消长过程中保持本身平衡。

如前所述,B-tree 常用的扩充形式是 B^+-tree。给定适当的 k 值,B^+-tree 有下述限制和规定。

(1) 每个结点最多有 $2k$ 个索引键值,索引键值也称为 B^+-tree 的秩。

(2) 根结点至少有一个索引键值,其他结点至少有 k 个索引键值。

(3) 非叶结点如果有 m 个索引键值,就有 $m+1$ 个子结点。

(4) 所有叶结点位于树的同一级上,树是平衡的。

G-tree 组成类似 B^+-tree 的结构,用于空间点对象的查询。

下面以二维空间为例说明 G-tree 原理。其方法可以推广到任意维空间。

1. G-tree 构造

G-tree 与 Kd-tree 类似,对数据空间采用按维数轮换交替对空间进行剖分,但不同的是采取平均剖分方式。为了突出原理,简化叙述,以下将所涉及的属性值都规范化为{0,1}中的值,同时要求剖分后每个区域中包含的数据点不能超过两个,如果超过两个,则按循环交

替的次序继续剖分空间，直至满足每个区域不超过两个点的要求为止。

在图 5-18(a)中，设平面上开始有 3 个点，按照 x 方向等分为 A、B 两个区域，A 中有两个点，B 中有一个点，满足要求。

如果在 A 中又插入两个点，由于 A 中有 4 个点，A 需要剖分。按照剖分法则，A 首先按照 y 方向等分为上下两区域，其中 C 中只有一个点，满足要求。但 A 的下半部仍有 3 个点，须进一步剖分。先按照 x 方向等分，其中右半部 F 为空，因此左半部须要进一步剖分为 D、E 两区域，其中 D 中有两个点，E 中有一个点，剖分完毕，如图 5-18(b)和(c)所示。

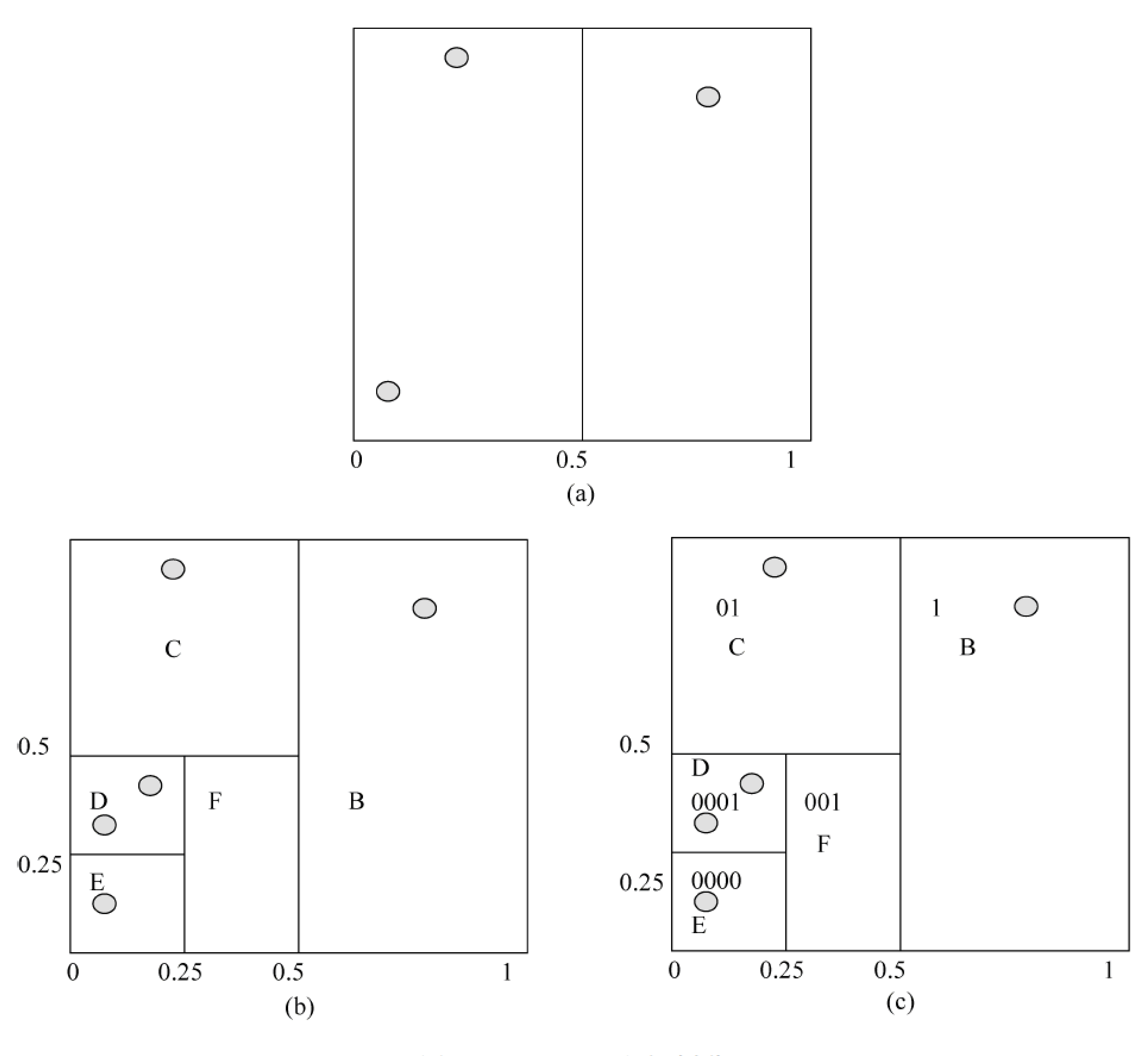


图 5-18　G-tree 空间剖分

图 5-19 表示各个区域的二进制编码，其编码规则与栅格的 Z 次序编码一致。为了清晰起见，略去其中的点。

G-tree 的这种空间剖分策略具有下述特点。

(1) 区域二进制编码构成全序。

(2) 区域集合构成平面的一个划分。

(3) 区域二进制编码的位数越多，则该区域越小，它是其编码前缀所代表的区域的子空间，如 0101 是 01 的子空间。

剖分区域由其编码组成类似于按 B^+-tree 的生成方式(由叶结点向上)构建 G-tree。在图 5-20 中，叶结点中每个索引项可用二元组(B，P)表示。B 代表区域的二进制码，P 是指

图 5-19　剖分区域编码过程

针，它指向一个页（块），页中存储该区域中点所对应的数据。例如，在多属性查询中，这些数据是指该区域中所有点所对应的元组集。这些元组应簇集存放。为了清晰起见，在图 5-20 中，每个结点最多只能有两个索引项，最少应该有一个索引项。在实际的 G-tree 中，每个结点是一页（块），其最大索引项的个数要远大于此数，但这种简化并不影响原理的说明。

图 5-20　G-tree 实例

2. G-tree 查询

G-tree 查询按下述两个步骤进行。

1）查询数据编码

在查询某点时，先将该点的坐标(x_q，y_q)转换成它所在区域的二进制编码。由于在查询前并不知道平面剖分情况，需要在 G-tree 中保留一个动态变化参数 L。L 表示 G-tree 中当前区域编码位数的最大值，如在图 5-19 中 $L=4$。因此，(x_q，y_q)所在的区域编码最多不超过 4 位，可以先按照 4 位生成。设待查点的坐标为(0.2，0.8)，则其所在区域的编码如表 5-1 所示，即其 4 位二进制编码为 0101。

表 5-1　(0.2，0.8)所在区域编码的生成

比　　较	0/1	编　　码
0.2＜0.5	0	0
0.8＞0.5	1	01
0.2＜0.25	0	010
0.8＞0.75	1	0101

2）使用编码前缀进行查询

从图 5-18(c)可以看出，图中并无编码为 0101 的区域。这表示没有剖分出编号为 0101 的独立小区域，它包含在其他区域中，包含它的编码可从 0101 的前缀中去寻找。显然，包含在编码为 01 的区域中，即 C 中。因为 C 没有再细分，所以 0101 后两位是多余的。在检索 G-tree 时，如果发现 G-tree 中的比较值等于生成编码的前缀，即以此前缀代替此生成编码进行检查。以图 5-20 为例，首先用 0101 对根结点的 3 个分支进行判别。01 取代 0101 进行比较，从而选中第二分支，由此追踪到第二叶结点。从索引项 01 的指针，可以查得 C 中所有点对应的数据。从图 5-18(b)可以看出，C 中并无(0.2,0.8)这样的点。因此，查询结果是无此点。

3. G-tree 更新

G-tree 更新包括插入更新和删除更新。

1）G-tree 插入

在插入时，如果插入后没有一个区域超过 2 点，则不需要调整 G-tree，否则，那些超过 2 点的区域需要分裂，而且这种分裂可能一直传播到根。设在图 5-18 中，插入一点(0.2,0.3)，按照图 5-19 的方式生成其所在区域编码为 0001，即区域 D。区域 D 中插入点(0.2,0.3)以后，域中增至 3 点，按分裂规则应按照 x 方向分裂为左右两个区域，其编码分别为 00010 和 00011，则图 5-20 中最左边的叶结点溢出，应分裂为两个叶结点，叶结点因而增加到 4 个。原来的根结点也要分裂，并产生新的根结点。G-tree 从两级增加为三级。插入后的 G-tree 如图 5-21 所示。

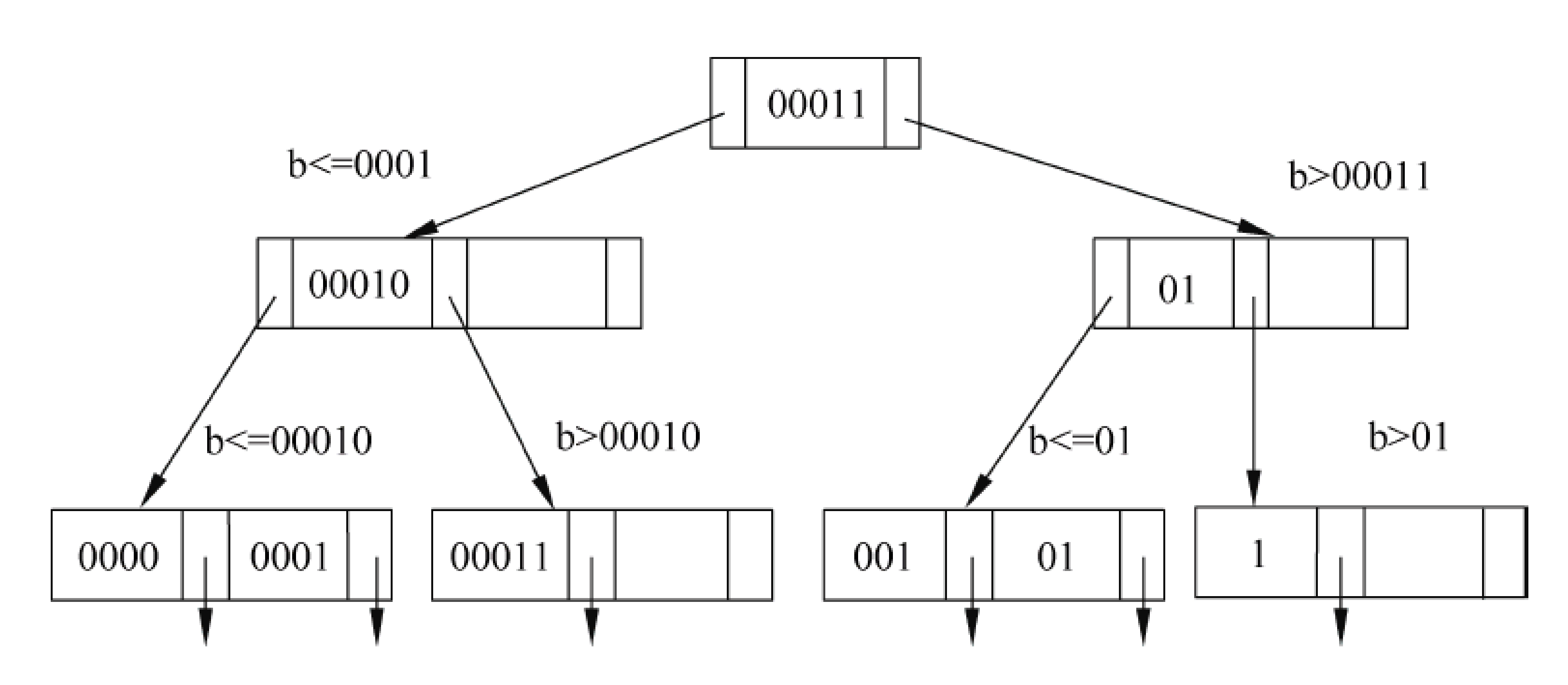


图 5-21　插入点(0.2,0.3)后的 G-tree

2）G-tree 删除

在删除时，由于 G-tree 允许空区域存在，因此，只须删除要删除的点，对 G-tree 本身一般不需要调整。如果出现两个空区域，其编码只是最后一位不同，即一个为 0，另一个为 1，这两个空区域可以合并成一个大的空区域。这样的合并工作，可以在删除时进行，也可以留待重构 G-tree 时进行。

5.6　空间区域索引技术

数据库系统主要是为处理海量数据而设计研发的，而数据处理的一个基本要求就是迅捷高效，这需要合理安排数据的物理存储结构，建立适当的索引技术。

如前所述，空间非点对象形状复杂，通常需要采用“近似”图形进行描述，即将索引结构按照一个或多个规则的更加简单的空间图形（称为空间码）来管理。其中一个最基本的情形就是使用前述的最小限定矩形 MBR（外包矩形），即内含涉及的空间对象并且边与坐标轴平行的最小矩形。

空间索引实质是在响应一项查询请求时，通过自身结构以获取该空间中所有对象的一个满足相应查询“必要条件”的关联子集，并返回该关联子集。

5.6.1 R-tree

下面讨论基于测度非零的空间图形的 R-tree 技术。

1. R-tree 构造

空间查询对象可以分为“大小”度量（测度）为零和不为零两种情形。测度非零空间对象一般用最小限定矩形 MBR 近似表示。R-tree 由 Guttmann 于 1984 年提出，可以看作是 B^+-tree 在高维情形下的推广。R-tree 以 MBR 为基本检索对象。与 B^+-tree 类似，R-tree 也是一种平衡多分树，但相对于 B^+-tree 而言，两者还具有下述差异。

（1）B^+-tree 通过逐级比较索引项中的索引值搜索查询对象；R-tree 是通过逐级缩小搜索空间范围来定位查询对象。

（2）B^+-tree 中的数据项具有顺序结构；R-tree 中的数据项没有顺序，而且可以相互重叠。

如前所述，当一个 MRB 包含有其他 MRB 时，就称其为目录 MRB，否则称为对象 MRB。

下面给出 R-tree 的基本描述。

给定适当的整数参数 m 和 M（$m<=M$），R-tree（Rectangle tree）是其结点满足如下约束条件的一种平衡多分树。

① 除了 R-tree 是只具有根结点的“根树”情形，其余所有 R-tree 根结点至少有两棵子树索引项。

② R-tree 的每个非叶结点包含有 $m \sim M$ 个索引项。

③ 在上述约束条件下，R-tree 保持树的平衡。

对于上述定义可以作如下说明。

（1）R-tree 中每个索引项为二元组（r，p），其中 r 为索引项表示的 MBR，p 为相应指针。对于叶结点，p 指向 r 所近似表示的空间数据对象。在非叶结点，p 指向 r 的子结点。

（2）R-tree 中一个结点最少有 m 个索引项，最多可有 M 个索引项。m 一般为 $2 \sim M/2$。由于 R-tree 在分裂时涉及区域的优化与调整，开销较大，为了减少分裂概率，m 宜选择适当数值。实验结果表明，m 在 $0.4M$ 左右可以获得较好的性能。

（3）R-tree 中一个对象 MBR 可位于多个不同目录 MBR（外包矩形）之中，此时，只需选取一个目录 MBR 包含该对象 MBR 即可。

图 5-22（a）表示了 MBR 的取法，5-22（b）是其对应 R-tree 构建。

2. R-tree 查询

给定查询矩形 $R_q=\{(x_1,y_1),(x_2,y_2)\}$，其中$(x_1,y_1)$、$(x_2,y_2)$分别为 R_q 的左下和右上角顶点坐标，为了叙述简便，以下假设需要查询 R_q 包含的空间对象。基于 R-tree 查询步骤可以表述如下。

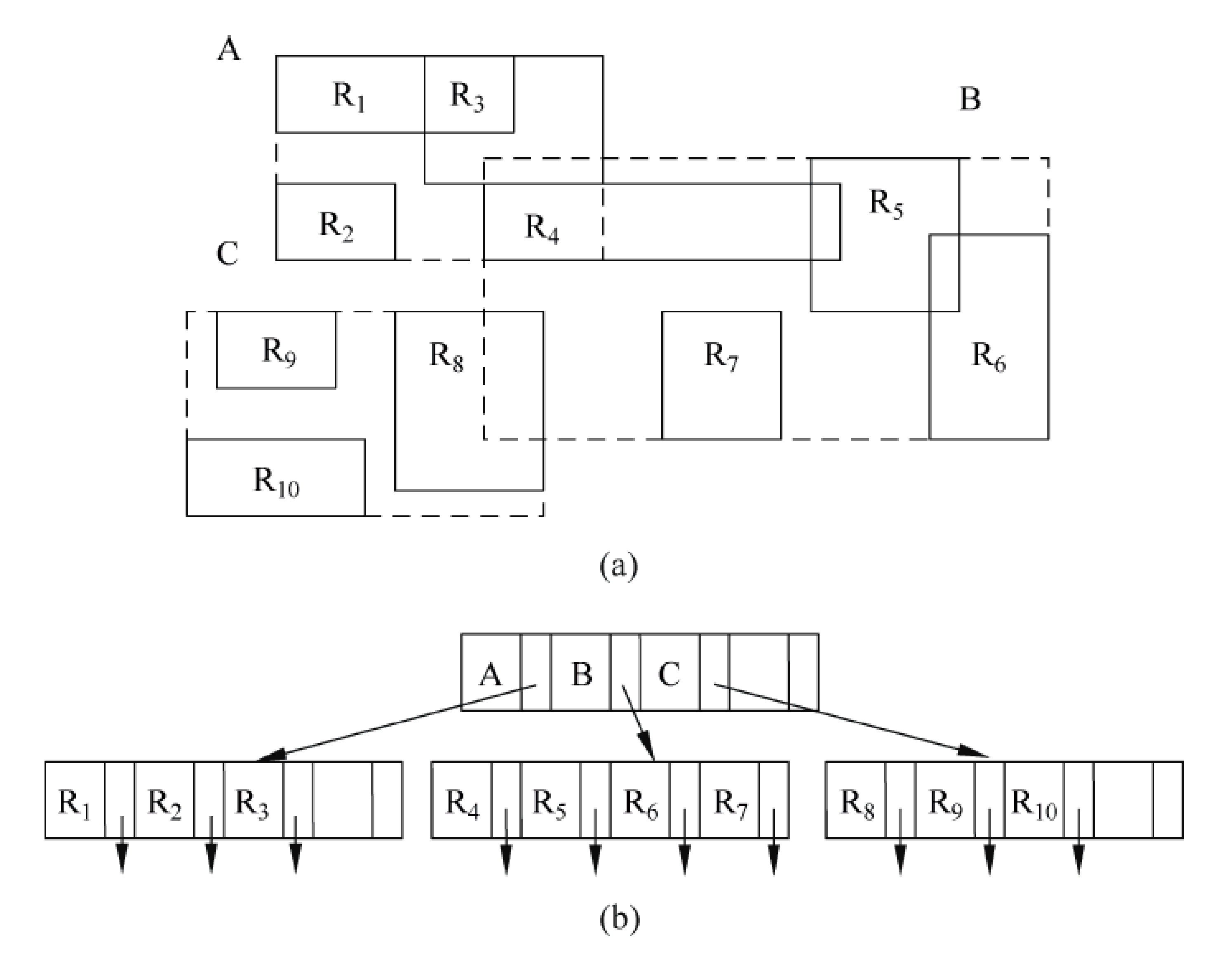


图 5-22　MBR 和 R-tree

(1) 由 R-tree 根结点开始：如果根结点中数据项本身就是对象 MBR，由于 MBR 的无序性，此时需逐个检查每个对象 MBR 是否包含在 R_q 中，如果包含在 R_q 中，则就是选中对象之一。设对象 MBR 为 $R_0=\{(x_3,y_3),(x_4,y_4)\}$，其中$(x_3,y_3)$、$(x_4,y_4)$分别为 R_0 的左下和右上角坐标。R_0 包含在 R_q 中条件为：

$$x_3 \geqslant x_1 \wedge y_3 \geqslant y_1 \wedge x_4 \leqslant x_2 \wedge y_4 \leqslant y_2$$

(2) 如果根结点中索引项是目录 MBR，需要逐个检查其中目录 MBR 是否与 R_q 相交。如果相交，则此目录 MBR 中的对象 MBR 就有可能包含在 R_q 中，沿此目录 MBR 继续搜索；如果目录 MBR 与 R_q 不相交，则其中对象 MBR 不可能包含在 R_q 中，不必沿此目录继续搜索。

(3) 当搜索到叶结点，与前述相同，需要逐个检查其中对象 MBR 是否包含在 R_q 中。需要注意的是，当目录 MBR 之间有重叠时，即使重叠区中对象只属于一个目录 MBR，但要判定其属于哪一个目录 MBR，仍然需要多路搜索。

3. R-tree 更新

作为 B^+-tree 的高维拓展，R-tree 的重要特色在于：插入更新和删除更新过程中需要保持树的平衡。

1) 插入更新

设(R_1,p)是待插入对象 MBR 及指向对象指针的二元组。由于空间对象本身无序，因此与 B^+-tree 不同，插入哪个叶结点及在叶结点中的次序不是唯一的。R_1 插入哪个叶结点关键在于是它属于哪个目录 MBR。

(1) 目录 MBR 的选择。实际问题中，R_1 可能与多个目录 MBR 相交或相邻，这些目录 MBR 只要其中的对象 MBR 不超过目录 MBR 中的最大值，经过必要扩大后，都可被选来覆盖 R_1。因此，R_1 插入方式可能有多种选择，一般来说，需要进行必要的优化。R-tree 采用较简单的优化准则，即选择插入后面积扩大最小的目录 MBR 去覆盖；若有两个或多个目录

MBR 都满足此要求,则选择其中面积之一即可。

(2) 叶结点的处理。目录 MBR 选定以后,就可在其子结点中确定插入对象所在的叶结点。如果叶结点中有空,则可直接插入,插入完成;如果叶结点在插入后溢出,则需要分裂为两个。由于叶结点中数据项无序,叶结点中数据项重新分组方式可有多种选择,这在许多 R-tree 改进方案中都有讨论。

(3) 分裂的传播。叶结点分裂后,需要生成两个目录 MBR,覆盖分裂后的两组对象 MBR,以取代原来的目录 MBR。这就意味着叶结点的上一级结点要增加一个索引项。这又可能导致该结点分裂,这种分裂可传播至根。根分裂后,树就要增加一级。

插入更新是 R-tree 相应数据操作中的最大开销项,对索引性能影响很大,因而是 R-tree 改进方案中的研究重点。

2) 删除更新

删除更新通常有下述步骤。

(1) 搜索删除。在删除某个对象 MBR 时,先按照其对象 MBR,从根搜索到它所在的叶结点,从叶结点中将其删除。

(2) 平衡调整。如果删除后叶结点中对象 MBR 的个数仍然大于最小值,则该叶结点保持不变。否则,在删除该对象之后,可能引起叶结点下溢(underflow)。在此情况下,可将此叶结点及其对应目录 MBR 删除,且将其中剩余的对象 MBR 及其指向对象 MBR 的指针重插入树中。这实际上相当于把删除了的叶结点中的剩余对象 MBR,分摊到与其相交或相邻的其他目录中。

(3) 下溢处理。目录对象 MBR 的删除也会引起其所在的非叶结点的下溢,从而导致该非叶结点的删除。其中,剩余的目录 MBR 对象可以重插到其他同级的非叶结点中。这个过程有可能继续传播到根结点的删除。在根结点删除时,如果其中只有一个索引项,则可以直接删除,无须重插入,此时 R-tree 降低一级。在重插入的过程中也可能引起结点的溢出,如除被删除的叶结点外,其他叶结点又分裂。通过这个过程,不但可以消除叶结点的下溢,而且在一定程度上改变了叶结点中索引项分配不均匀的情况。这在效果上相当于对 R-tree 的一次合理重组。

R-tree 是最早提出的大小非零空间对象的索引结构。由于 R-tree 具有 B^+-tree 的平衡性特点,适用于多维空间对象的检索。它一经提出就受到广泛关注,在 SDBS 中得到较广的应用。针对 R-tree 的不足,人们也提出了不少 R-tree 变种,目标主要集中在 R-tree 更新性能的改善上。

5.6.2 R^*-tree

与 B^+-tree 情形类似,在数据动态管理过程当中保持索引树的平衡性是 R-tree 的关键所在。其中,结点插入算法是更新过程中的基本技术,而在删除过程中出现的结点下溢引发的结点重组从某种意义上也可以看作是一种结点插入过程。

B^+-tree 中的数据项按照键值有序,因此数据插入是按照所给定的顺序插入到唯一确定的位置。但 R-tree 中 MBR 难以定义顺序,取定其插入位置只能采用循照树中父/子结点路径的方式确定而难以有其他选择的可能。由于 R-tree 中 MBR 的重叠性质,存在着相应的多条确定位置的插入路径,因此也就具有多个不同的同级插入位置。这样,相对

于 B^+-tree，更为复杂的 MBR 重叠性反而为寻找“更优”的插入位置提供了可能。Beckmann 等人正确利用了 R-tree 的“重叠”特征，在 1990 年提出了基于插入优化的 R-tree 改进方案，也就是著名的 R^*-tree。

R^*-tree 继承了 R-tree 的基本结构，突出之处就是针对 R-tree 自身 MBR 可能重叠的特征对相应插入算法进行了优化改进，显著提高了索引树的基本性能，其中的分析设计相当精细也非常精彩。

从逻辑表述角度考虑，R^*-tree 的这种优化改进可以分为：“插入结点过程中的优化策略”和“删除过程中结点强行重新插入和结点分裂优化”两种情形。

1. 插入结点优化策略

在插入位置的多路选择过程中，R-tree 中对象 MBR 所需插入目录 MBR 的选取着眼点为是否遵循“最小性原则”。

最小性原则：在一个结点中插入新的对象 MBR 后，所有包含该结点的目录 MBR 都可能需要进行“扩大”，其“扩大”程度取决于新插入的对象 MBR。常规 R-tree 选择使得其目录 MBR“扩大”最小的结点作为宿主而将新的结点插入其中。

与 B^+-tree 相同，R-tree 中的非叶结点中的索引键值实际上只起到导航作用，真正需要查询的数据都存储在叶结点当中。R-tree 中同级结点对应的目录 MBR 都可能存在着重叠，并且各个 MBR 中也存在着大小不等的未含子结点 MBR 的空白空间即“死空间”。这种重叠和死空间对于非叶结点查询处理影响不大，但对在叶结点中进行实际对象 MBR 的查询性能却具有很大影响，需要进行精心处理。

这里问题的要害就在于 R-tree 对非叶结点和叶结点采用同一种插入准则而没有充分考虑到前者“导航查找”和后者“数据查找”的不同查询特性。

针对这种情形，R^*-tree 采取了下述插入结点优化策略。

(1) 对于非叶结点中的索引键值插入：依然采用 R-tree 的“最小性原则”。

(2) 对于叶结点中的实际数据插入：选择插入后使其目录 MBR 与其他目录 MBR 重叠面积之和最小的叶结点。

(3) 对于有多个叶结点满足“(2)”的情形：对这些叶结点采用 R-tree“最小性原则”，即选择其中目录 MBR 在插入后面积扩大最小的叶结点。

按上述优化准则选定当前插入结点后，如果当前结点为空，则将索引项直接插入到该结点中。如果插入当前结点后发生溢出，R^*-tree 采用与 R-tree 不同的溢出方法，不是立即分裂结点，而是根据实际情形设法避免分裂。而在必须分裂结点时，R^*-tree 也提供较好的分裂优化策略。

2. 重新插入与分裂优化策略

结点溢出后，需要进行结点重组，即或者重新构建一个新结点，或者将原来结点中的部分结点插入到同级的其他结点当中。

在 B-tree 中，结点按照数据键值有序，同样一批数据，当按照不同工作顺序插入到索引树中时，由于需要按照自身的键值顺序进入到确定的插入位置，插入先后的次序不会对索引性能产生大的影响。但 R-tree 就有所不同，由于其插入位置的多路可选择性，如何组建新的目录 MBR 就与对象 MBR 插入先后次序产生很大关联。同样一批对象 MBR 按照不同的进入次序可以组建完全不同的 R-tree，这些 R-tree 彼此之间具有相当大的性能差异。在

R-tree 需要分裂的情况下对其进行适当的“局部”调整，这就相对于常规情形增加了索引树的更新开销，但从数据查询的快速响应性考虑，这种“额外”开销应当是人们可以接受的，毕竟数据库主要是用来查询的，其更新工作作为查询的“前期”开销尽管加大了，但对于“当前”的查询效率却得到了较大提升。

R^*-tree 正是从上述分析出发，将 R-tree 只在下溢出时采用的方法推广到插入而产生结点溢出引发分裂的情形，采取了不同于 R-tree 简单重组的优化改进策略。

1）重新插入的基本思想

当结点发生溢出时，R^*-tree 并不立即分裂该结点，而是从溢出结点中抽取出约 $\rho\times M$ 个被索引的 MBR 重新插入到其他同级结点中，其中 ρ 是可通过实验确定的小于 1 的正数，通常是取在 0.3 左右。

为从溢出结点中选取 $\rho\times M$ 个被索引 MBR，可将溢出结点中各 MBR 按其图形中心与该结点 MBR 中心的距离降序排列。首先选取处于溢出结点 MBR 边缘上的被索引 MBR 进行重插入，这是由于边缘上的被索引 MBR 在重插入时有可能插入邻近的结点 MBR 当中。当 $\rho\times M$ 个索引项中有一个成功地插入到同级的其他结点中，则该结点就可免除分裂。

按照相关统计，在常规情况下，R^*-tree 结点平均装满程度约为 70%。从溢出结点中抽取出大约 30%的被索引 MBR，逐个找到比较合理的同级结点位置，从而可以部分地纠正由于随机插入 MBR 所引起的数据在索引中的不合理分布。实际上，重新插入到其他同级结点后，其装满程度可降至这一统计平均值，因此成为 R^*-tree 提高性能的有效措施。

由于索引树特性，并不是对于所有情形都能实施重新插入。

(1) 重新插入前提：出现溢出的结点应该具有相应的同级结点，而根结点的唯一性决定其没有任何同级结点，因此，对于根结点的溢出情形不能实施重新插入。

(2) 避免出现循环：对于 R^*-tree 中除了根结点的其他各级结点溢出情形都可首先考虑使用重新插入而不是立即分裂。但重新插入可能引起其他同级结点溢出，由此引发新的重新插入，如此下去也许会无限循环。为避免这种情况发生，R^*-tree 规定，在同一级结点中只有当发生第一次溢出时才能施行重新插入。

(3) 不能重插入情况处理：$\rho\times M$ 个被索引 MBR 在重新插入时有可能仍然插回到初始结点，由此导致重新插入失败。对于不能实施重新插入策略的结点，就只能进行结点分裂，但 R^*-tree 中的结点分裂相对于 R-tree 情形也有着自己独特的优化改进策略。

2）必须分裂结点的优化策略

由于不能在所有溢出情形下都使用重新插入策略，因此 R^*-tree 也存在着需要对结点进行分裂的情况。R^*-tree 对必须实行的结点分裂采用了适当优化改进技术。

当一个需要分裂的结点应该具有 $M+1$ 个被索引项，而在分裂成两个结点后每个结点至少应该具有 m 个被索引项，最多可以包含 $M+1-m$ 个被索引项。此时，$M+1$ 个被索引项需要依据各自空间位置分为两组以形成两个新的同级结点。相应具有 $(M+1-m)-m+1=M-2m+2$ 种分组方式。为了叙述简单起见，以下只对二维情形说明分裂结点优化策略。

(1) 对被索引项进行排序。将这 $M+1$ 个被索引项按照 MBR 的空间特征进行排序。

此时，可以分别存在按照被索引项 MBR 的 x 坐标的最小和最大值、y 坐标的最小和最大值进行的 4 种排序方式。对于如图 5-22(a)所示的目录 MBR 中的 B 而言，相应的 4 种排序结果述如下。

① 按照 x 坐标最小值排序：$<R_4, R_7, R_5, R_6>$。

② 按照 x 坐标最大值排序：$<R_7, R_4, R_5, R_6>$。

③ 按照 y 坐标最小值排序：$<R_7, R_6, R_5, R_4>$。

④ 按照 y 坐标最大值排序：$<R_7, R_6, R_4, R_5>$。

(2) 将排序后被索引项分为两组。其中第一组选取已排序列前面若干项，而第二组就选取后面的剩余项。由于每组被索引项的个数需要在 m 和 M 之间，因此对于已排序列中的 $M+1$ 个被索引项来说，就有 $(M+1-m)-m+1=M-2(m-1)$ 种分为两组的方式。所有可能的两组分割方式中各组包含被索引项情形如表 5-2 所示。

表 5-2　$M+1$ 个被索引项可能的两组划分

组　别	方式 1	方式 2	…	方式 $M-2m+1$	方式 $M-2(m-1)$
第一组	m	$m+1$	…	$M-m$	$M+1-m$
第二组	$M+1-m$	$M-m$	…	$m+1$	m

每种排序有 $M-2(m-1)$ 种两组划分方式，任取 $k(1\leqslant k\leqslant M-2(m-1))$，选定第 k 种分组方式分别对应的覆盖第一组和第二组中被索引项的目录 MBR 并记为 MBR_{k1} 和 MBR_{k2}，两者的周长分别记为 c_{k1} 和 c_{k2}，计算出$(c_{k1}+c_{k2})$并对于所有满足 $1\leqslant k\leqslant M-2(m-1)$的 k 做和 $\sum(c_{k1}+c_{k2})$。

(3) 选择排序方式。对于上述 4 种排序方式，可以分别得到 4 个 $\sum(c_{k1}+c_{k2})$ 的值，然后选取其中 $\sum(c_{k1}+c_{k2})$ 值最小的一种排序。在前述 4 种排序序列中，都需要对相同的 $M+1$ 个被索引项进行二组划分，当 $\sum(c_{k1}+c_{k2})$ 最小时就表明此种排序分组中，各组对应的 MBR_{k1} 和 MBR_{k2} 最接近于正方形。而在周长一定的情况下，只有正方形的面积最小，有利于高一级目录 MBR 的组建和减少其重叠，符合常规 R-tree 的最小性原则。

(4) 确定分裂后的新结点。选定了适当的排序方式后，就需要在基于该排序的 $M-2(m-1)$ 个二组划分中选取一种分组，这就可以按照 MBR_{k1} 和 MBR_{k2} 重叠面积最小的原则进行确定。在存在多个分组都具有相同的最小重叠面积情况下，可以在按照面积之和进行选取。

(5) 取代溢出结点。最终确定分组之后，就以相应目录 MBR 取代原先发生溢出的结点，同时还需在发生溢出结点的父结点中以两个新的结点取代溢出结点。如果取代后该父结点也发生溢出，则继续按照“首先重新插入，否则分裂优化”的原则进行处理。注意到这种溢出可能一直传播到根结点。如前所述，对根结点只能进行分裂处理。

由上述可知，R^*-tree 是在 R-tree 的基础上优化改进了结点的重插入方法，尽量避免结点的分裂，在必须分裂结点时提供了优化分裂的方法。使目录间的重叠最小，在查询时减少了 I/O 操作，提高了查询效率。

本 章 小 结

SDB是在地理信息系统(GIS)基础上发展起来一门前沿交叉学科,也是高级数据库技术中的基本领域之一。空间数据是指与客观对象的空间位置特别是地理位置有关的数据信息,而这种信息在实际应用中占有相当大的比例,据统计可以达到80%。与一般数据相比,空间数据具有数据量巨大、数据结构复杂和数据操作计算密集多样等特点。随着信息技术的发展进步,现代社会对位置服务和分析决策的需求日益迫切,研究和掌握空间信息技术理论与方法的重要性也就日益显现。

空间数据显著特征是其几何形状和相互关系,与常规数据相比具有较大差异。在SDB中,首先将其抽象化,建立起“点”“线”和“区域”等基本数据类型,以及“划分”和“网络”等导出数据类型。空间数据反映空间形体,研究空间形体的基本着重点还在于形体相互之间的关系,包括度量关系、方位关系和拓扑关系。其中,拓扑关系是空间数据管理的基本点之一。

SDB源于GIS,而GIS中有基于空间坐标(空间位置)的绝对空间和基于属性特征(空间关系)的相对空间之分,通常就以此建立两种不同的空间数据模型,即基于相对空间概念的镶嵌模型和基于绝对空间概念的矢量模型。

一般图形的形状在数学上难以描述,相应的拓扑关系也相当复杂,在计算机中就更不易表示、存储和处理“完全真实”的空间形体。在SDB中,通常使用某种简单的最小外接图形“近似”相应的实际空间对象,这就是SDB中独具特色的空间对象近似技术。通常采用相邻边平行于坐标轴的最小限定矩形MBR。由此就可以建立起空间对象数据表示、存储和处理的一套有效机制。

引入了空间近似技术之后,相应空间对象之间的关系就转换为相应MBR之间的关系,从而为计算机处理提供了可能。但MBR之间关系只是实际空间对象之间的一种“必要关系”而非“充分关系”。例如,两个MBR不交,相应空间对象必定不交,但MBR相交并不说明对象空间对象也一定相交。因此,使用空间近似只是缩小了数据查找范围。注意到数据索引本质就是利用“必要”条件缩小查找范围,因此空间近似实际上就为空间数据索引开辟了技术支撑的通道。

R-tree索引就是建立空间近似基础上的一种空间数据索引技术。从“平衡树”的角度,R-tree可以看作是一维B-tree在多维度上的推广,现已成为空间数据管理的经典技术,也被有效应用在各类涉及“广义”维度处理的问题当中。例如,在时态数据管理当中,将“有效时间”和“事务时间”看作两个广义维度,进而使用R-tree研究相应的双时态索引。R-tree和B^+-tree已经成为数据库索引技术中的两个重要的标志性索引技术。

R-tree的一个基本特征是下层中的一个结点项可以包含在上一层不同的索引项当中,也就是说,到达一个结点可以有多条不同的索引路径,这就是R-tree的结点重叠问题。结点重叠为数据插入中多条插入路径选择提出了挑战,而作为R-tree解决此问题的拓展,R^*-tree修改插入和分裂算法并引入强制重新插入机制,它不同于R-tree只考虑目录MBR面积大小,R^*-tree在选择插入路径时同时考虑目录MBR面积、空白区域(死空间)和重叠区域大小。在学习时,应当注意领会和理解R^*-tree构建过程中对R-tree特征的精细分析和精致处理。实际上,只有掌握了R^*-tree的基本思想和处理技术,才会对R-tree索引技术系

列具有比较深刻的了解与体会。从索引效率角度来看，由 R-tree 到 R^*-tree 可以类比于由 B-tree 到 B^+-tree。

主要参考文献

[1] 陈国平.空间数据库技术应用[M].武汉：武汉大学出版社，2013.

[2] 王能斌.数据库系统教程(下册)[M].北京：电子工业出版社，2002.

[3] 吴信才.空间数据库[M].北京：科学出版社，2016.

[4] 毕硕本.空间数据库教程[M].北京：科学出版社，2016.

[5] 郭薇，郭菁，胡志勇.空间数据库索引技术[M].上海：上海交通大学出版社，2006.

第 6 章　时态数据库

时间是自然界无处不在的客观属性。现实世界是一个四维时空，从本质上来看，三维空间中任何客观对象都是动态变化的，而变动是一个必须用时间方能刻画的概念。因此，发生在现实世界中的客观对象都带有时间的烙印，事物的各类特征信息与相应时间信息的关联在表述和研究客观对象过程中发挥着基本的作用。在数据库和以数据库为核心的各类信息系统中，有效管理各种基于时间的数据信息是一个在原理研究方面不断深化和在技术方法方面逐步创新的自然进程。传统数据库把时间看作一般属性用于描述客观对象发展过程中的特征，没有考虑不同时间状态下数据的相互关联及相应处理过程中的时间约束，难以有效表示和处理时间应用环境下的各类数据信息，时态数据库（Temporal DataBase，TDB）技术由此应运而生。本章首先引入时间数据结构、时间数据类型和建立其上的时间运算，这是 TDB 中处理时间相关数据的基础；然后介绍基于 RDB 的时态数据模型；最后简要叙述双时态数据库操作语言 ATSQL。

6.1　时间与时态数据库

处理时间相关数据信息，首先需要明确时间概念并从计算机技术角度建立时间模型，讨论相应的时间操作，这是 TDB 的必备基础；然后需要根据计算机处理时间的不同角度建立不同类型的 TDB。

6.1.1　时间基本概念

时间现象通常可以通过事物变化演进的过程和结果为人们所感知，但对于什么是“时间”至今仍然是很多领域特别是哲学领域和物理学探讨的重要课题。

1. 哲学领域

哲学研究各类本体问题，而此时人就具有两个层面上的含义，首先，从整体上而言，人和世界上万事万物一样都是客体，都可以作为基于客体的本体进行探究；但人又是研究所有客体的本体含义的“始动者”，这种被研究对象和研究者的双重身份实际上构成了众多哲学（当然也有科学）难题的底层支撑，“时间”概念也是如此。近代哲学家们看来，时间是与时间研究者和使用者密切相关的。其中，胡塞尔和海德格尔师徒二人的观点具有代表性。

1）胡塞尔的时间观

胡塞尔现象学有两个基本观点：一是本质直观方法；二是先验还原方法。后者着重讨论“存在”问题，其中一个重要概念就是纯粹意识。胡塞尔认为纯粹意识由印象、回忆和展望

构成，而这些必然就隐含着现在、过去和未来的时间概念，也就是说，纯粹意识实际上就构成了人们整个内在的时间意识。

2）海德格尔的时间观

作为胡塞尔的高徒，海德格尔实际上是使用现象学方法创建了哲学意义上的“人类学”，其中的关键元语就是“此在”（dasein）。他认为人的存在实际上就是打发掉一段时间，人的存在具有3种不同的方式：沉沦态、搁置态和生存态，它们分别对应于时间的过去、现在和将来。这三态实际上就是“此在”在时间中的生存方式。如果说“存在”是海德格尔哲学的核心，则时间就是存在的核心。

2. 科学领域

科学领域中的时间（当然还有空间）概念更多具有客观考量和技术操作的意义，这是与哲学中时间探究的不同之处。其中，牛顿和爱因斯坦的时间观已经成为当今科学技术领域中的经典。

（1）牛顿时间观：时间维与空间维相似但独立于空间维之外。空间和时间都是客观的、绝对的和独立存在的，与人们的感知与感受没有关系，并且空间和时间彼此无关。

（2）爱因斯坦时间观：客观世界是由一个时间维度和3个空间维度组成的四维时空，时间和空间彼此融合并且可以相互作用转换，没有独立于空间的时间，也没有脱离于时间的空间，时间与空间相结合而成为时空统一体。在狭义相对论中，时间和空间坐标像两个空间坐标一样并没有本质意义上的差别；在广义相对论中，时间和空间一样具有“双向性”，能够像进行空间旅行那样进行时间旅行。

3. 时间概念的复杂性

由于时间概念相当复杂，涉及人们所处环境的根本所在，很多问题都需要进一步探究，因此上述时间观实际上还没有真正统一人们对于时间的理解与把握。按照AL-Taha（1992年）的观点，产生时间概念复杂性的原因可能在于下述几个方面。

（1）时间语义多样性。一个时间用语可以有多种基于应用场景的时间语义的解释。例如，在英语中，Now可以指今天、这一个星期、今年或本世纪；Today可以指现在或今天；Future可以指明天、一周后或一年后；Past可以指一个小时前、昨天或一周前。

（2）时间粒度复杂性。对不同研究和应用领域，对时间理解与要求可以很不一样。对历史研究而言，一年是一个很小的单位；而对基本粒子研究来说，一秒就是一个很大的单位。

（3）事件过程多样性。人们通常是通过所从事活动的事件过程来理解把握时间，如“一顿饭的工夫”“月圆时分”等。但事件过程本身却非常复杂多样，如可分为外部的和内部的（人工）事件。外部事件是自然过程，如地震和海啸等；内部事件是非自然过程，如人类的战争和法律的颁布实施等。这种多样性为统一认识和理解时间带来了困难。

（4）时间感受的主观性。时间与人们的感受密切相关，在不同环境中的不同经历可以使人们具有完全不同的时间感受。爱因斯坦就曾经说过，当人在一个灼热火炉旁时会感觉到时间过得真慢；当和一个美丽姑娘待在一起时又会感觉到时间过得真快。此外，还有时间关于运动速度的相对性，如著名的“双胞胎详谬”。

在日常生活中，时间似乎表现得普通而简单，但对于计算机而言，进入到机器中的一切数据都需要严格界定和清晰描述。然而，对时间进行明确表述和科学模拟却是一项极具挑

战性的工作,人们为此努力探索了好几百年。也许人们事实上并不能在科学的意义上明确界定时间,而只能从事物的发展变化中来感知时间。从深层感知的语义上考虑,时间就是事件演进的序列,时间本质就是客观对象从一种状态变化到另一种状态的变化。客观事物变化有本质和非本质变化之分,前者导致新的客观事物出现,即产生一个新的对象;后者导致客观事物进化的一个新版本诞生。

人们或许可以通过状态(state)、事件(event)和证据(evidence)这三类数据来表述变化和感知时间。

① 状态:表达事物在某一时刻的属性。

② 事件:实体从一种状态转化为另一种状态,需要探讨和描述变化发生原因。

③ 证据:表明某个事件已经发生,某个实体已经从一个状态转化为另一个状态。

6.1.2 时间的数据结构

时间现象可以通过事物变化演进的结果为人们所感知,这可以看作是对时间内涵的一种哲学释义。事实上,对于"时间"而言,人们只能和对待"数据"和"信息"等"元"概念一样,在本体探究上保持敬畏,在特定的技术领域中采取"实用",即只对其(也只能对其)进行某种描述性概括。对于 TDB 技术而言,则主要从计算机应用与实现角度讨论时间表示及其数据结构。

1. 时间数据结构

从一般意义上考虑,人们理解和处理时间的基本单位是"时刻"或"时间点"。在实际应用中,通常是需要同时考虑多个相互具有语义关联的时间点组成的集合。因此,需要建立起时间(点)的数据结构。

1) 连续、稠密和离散数据结构

可以从最一般的时间关联角度来考虑连续、稠密和离散这 3 种时间数据结构。

(1) 连续时间结构:将时间点表示为非负实数,从而将所涉及时间集合建模为非负实数集合。通过对时间进行采样方式处理实数的连续性。此时,可以认为每一个非负实数都对应一个"时间点"

(2) 稠密数据结构:将时间点表示为非负有理数,从而将时间建模为非负有理数的集合。由于有理数的稠密性,通过在任意两个不同有理时间点之间都可找到另一有理时间点来解释有理数的稠密性。对于时间的连续和稠密数据结构,可通过适当差值方法以实现对于时间数据的计算机处理。

(3) 离散数据结构:将时间点表示为非负整数数,从而将时间建模为非负整数集合。由于非负整数的离散性,因此不能无限制地使用插值方法,否则就可能得不到任何离散时间点。此时,每一个非负整数都对应一个"最小"的不能再分的原子时间间隔,这就是建立在时间基元之上的时间粒度。

2) 线性、分枝和周期时间数据结构

按照人们直观经验,时间如同流水一样单向前进。从计算机数据处理的技术角度考虑,根据时间前进趋向的发展结果,可以建立时间的线性、分枝和周期数据结构

(1) 线性时间结构。时间从过去开始经过现在并按照顺序发展到未来。这种由过去到现在,再由现在到未来的进程具有线性递增特征,所有发展结果能够按照全序进行组织,构

成一种线性时间结构，如图 6-1 所示。例如，一个人从出生到离世过程中所经历的各种重要事件描述。

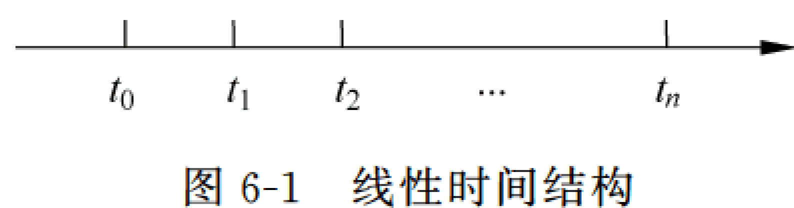


图 6-1　线性时间结构

（2）分枝时间结构。时间由开始到未来可能沿着多个方向发展，进程中的发展结果可以按照偏序进行组织进而形成一个分枝（树形）时间结构。分枝时间结构中任意两个时间结点不一定可进行比较。例如，在软件开发过程中，一个版本发展成不同的分枝版本，而分枝版本又会进行分枝演化，不同分枝版本上的内容结点通常不能进行比较。又如，人类发展史上由阿法南猿到包氏古猿再到能人，然后能人分别演化为尼安德特人、霍比特人、北京猿人和智人等多个分枝。分枝时间结构又可分为两种情形：一种是由过去到现在线性演进，而由现在到未来则有多种发展可能，如图 6-2(a)所示；另一种是从过去到现在有多种演进方向，但由现在到未来却是线性发展，如图 6-2(b)所示。

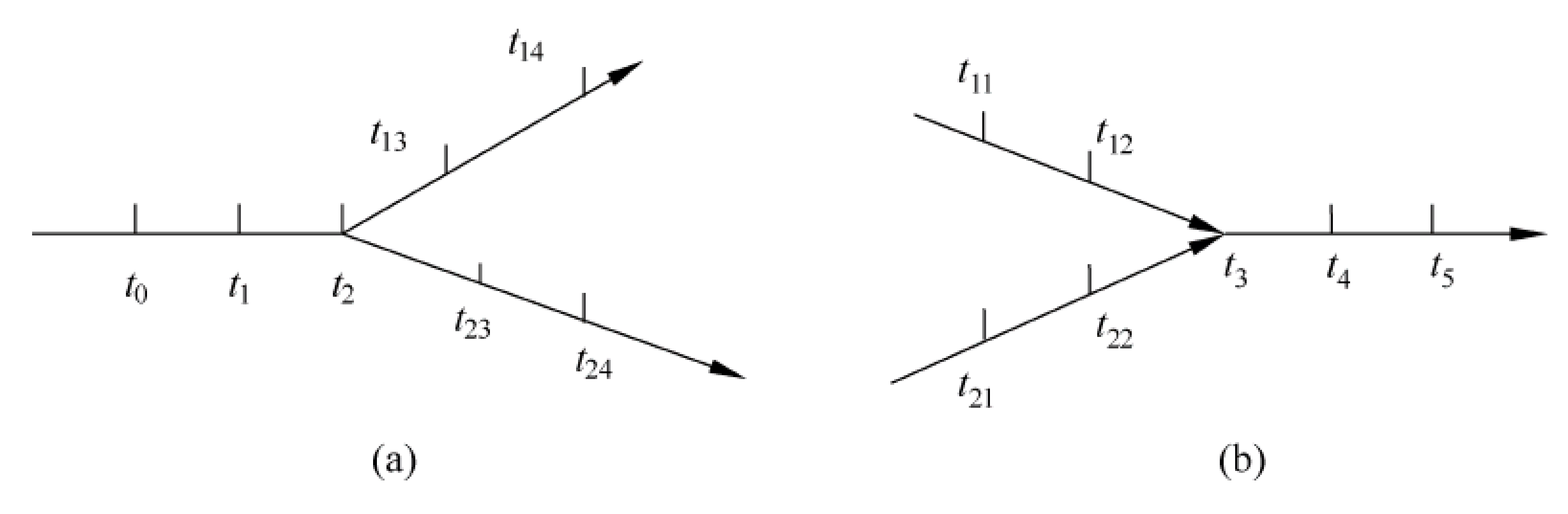


图 6-2　分枝时间结构

（3）循环时间结构。客观实体随时间由过去到现在再到未来周而复始出现循环，即具有某种周期性，此时所有发展结果构成一种图形结构，如图 6-3 所示。但从发展结果上来看，是难以区分“过去”和“将来”时间的。例如，钟摆的运动进程、随机投掷的硬币正反面的显示等。

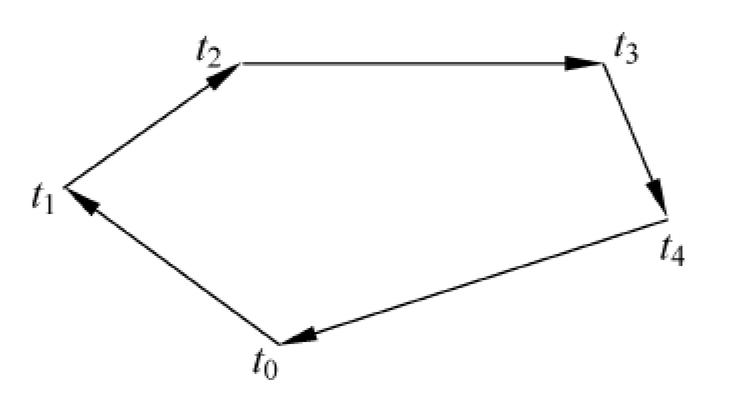


图 6-3　循环时间结构

对于时间数据而言，线性、分枝和周期结构实际上就是一般数据结构中的线性结构、树形结构和图结构在实际数据领域中的具体表现。在 TDB 中，主要讨论和采用基于离散类型的线性时间数据结构。

2. 时间基元与时间粒度

在实际应用中，所采用的时间轴线通常由一系列离散时间点（Time Point）也就是非负整数点构成，为了明确具体应用环境中的这些整数点的语义，还需要引入“时间基元”和“时间粒度”两个基本概念。

1）时间基元

从技术上来说，时间属于度量范畴。需要结合计算机自身特点确定一个时间的基准，这个基准包括时间的一个最小的不可分割的计时单位 C——这通常称为“时间基元”（Chronon）和一个时间的始点。

在计算机中，时间基元是由计算机芯片晶振频率决定的，一般取为 1/18.2 秒。

计算机中时间起始的基准点是 UMT 时间即世界标准时间(格林尼治时间)GMT:1970 年 1 月 1 日 0 时 0 分 0 秒。实时系统一般要求时间精度达到 10^{-3}～10^{-5}秒。

时间基元是计算机系统所支持的最小时间基准,在计算机内部以时间基元作为计时基准保证了计算机时间标准的唯一性,不同时区的不同时间表示只是为方便用户而实行的必要转换。

2) 时间粒度

在一项特定实际应用中所采用的描述时间数据的时间单位称为“时间粒度”(time granularity)。时间粒度表示时间点间离散化程度,其大小受到时间基元约束。作为计算机系统最小时间基准,时间基元是一个系统概念;而作为一个技术概念和用户应用时最小的时间单位,时间粒度不能小于时间基元。

时间粒度在某种意义下也可看作是时间基元的“倍数”,但在语义上和人们使用的时间单位关联。时间粒度反映了具体时态信息系统中描述时间点的各类适用单位。时间粒度越小,离散时间点就越多,描述事件变化的信息越精细准确。但太小的粒度会导致系统资源开销增加;反之,描述事件变化的信息越粗糙。

3) 单粒度和多粒度

在数据管理过程中,时间数据语义复杂性主要表现在时间粒度的多样性,因此,相应数据库系统中存在单粒度和多粒度两种形式。只支持一种时间粒度的数据库为单时间粒度系统,否则就为多时间粒度系统。对于多时间粒度系统来说,其中不同的数据库应用可使用不同时间粒度,如描述农业产量的数据使用“年”;描述工资信息使用“月”;描述某些科学系统信息粒度可以更小,取秒、微妙、毫秒等。

在实际应用中,选择粒度应根据实际需求和系统承受能力决定,而各种时间粒度相互间通畅转化则是系统处理时间数据效能优劣的基本标志之一。

3. 时间数据类型

在线性离散模型中,时间的语义与所采用的时间粒度相关。类似于程序设计语言中基本数据类型概念,在实际应用当中,还需要建立当确定时间粒度后的各种时间数据类型。

(1) 时间点(instant)。连续模型中时间点是时间轴上的实数点,离散模型中时间点是时间轴上的一个原子时间间隔,此时语义由相应时间粒度确定。例如,当时间粒度为“天”时,2016 年 12 月 1 日是时间点;而当时间粒度是“秒”时,上述时间点就由系统自动换算为 2016 年 12 月 1 日 0 时 0 分 0 秒。

(2) 时间期间(period)。给定两个时间点 t_1 和 t_2($t_1 \leqslant t_2$),以 t_1 为始点和以 t_2 为终点的时间期间$[t_1, t_2)$定义为集合$\{t |\ t$ 为时间点且$t_1 \leqslant t < t_2\}$。时间点可看作始点和终点重合的时间期间,理解为延续时间为 0 的一段时间。实际应用中,由于需考虑时间期间兼容时间点的表示和时间期间关系的关系运算,一般采用始点封闭、终点开放的“左闭右开”形式。

(3) 时间元素(periods)。有限个时间期间(时间点)的集合。时间元素在英文中也可表述为 Time Element。时间元素主要用于描述应用中的复杂时间数据。

时间期间是在时态数据管理过程当中的基本时间标签;时间点可以看作是始点和终点重合的时间期间;时间元素本身就是时间期间(时间点)集合。

(4) 时间区间(interval)。持续的一段时间的长度。例如,“1 year 3 month”“30days”和“28hours”等。通常选用适当时间粒度后多使用整数表示时间区间。时间区间也称为时间

跨度(time span)。

(5) 时间戳(timestamp)。由一些字符构成的序列,用于唯一地标识某一刻的时间。对于计算机系统而言,时间戳是指格林尼治时间 1970 年 01 月 01 日 00 时 00 分 00 秒(北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒)起直至当前时刻总的秒数。数据库系统中时间戳是系统中自动生成的唯一的二进制数,存储大小为 8 个字节。时间戳通常用作给关系表或元组行添加版本戳记。现有的数字时间戳技术则是数字签名技术一种应用变形,因此,时间戳可以理解为证明时间的可查证印章。

6.1.3 时间运算

数据库系统对时间数据进行有效管理需要具有相应的时间数据操作功能,而这就需要从原理上讨论时间数据的各类运算。类似于高级程序设计语言情形,在建立基本时间数据类型之后就需要讨论相应的时间数据运算。时间数据运算主要分为:时间算术运算和时间关系运算两种类型。

1. 时间算术运算

时间算术运算包括同种数据类型内和不同数据类型间运算。以下记 $P=[P_s,P_e)$为以时间点 P_s 为始点,且以时间点 P_e 为终点的时间期间。

1) 时间点与时间跨度算术运算

设 p、p_1、p_2 是时间点,len 是时间跨度。

(1) 时间点和时间跨度加法运算。时间点 p+时间跨度 len =新时间点 p。

“+”语义为:时间轴上时间点 p 向左平移了 len 个单位。

“+”满足交换律:p+len=len+p。

(2) 时间点和时间跨度减法运算。时间点 p-时间跨度 len=新时间点 p。

“-”语义为:时间轴上时间点 p 向右平移 len 个时间单位。

(3) 时间点和时间点减法运算。当 $p_1 \leqslant p_2$ 时,时间点 p_2-时间点 p_1=时间跨度。

此时运算语义为:时间点 p_1 和时间点 p_2 之间的距离(跨度),记为 p_2-p_1。

2) 时间期间与时间跨度间运算

$P=[P_s,P_e)$、$P_1=[P_{1s},P_{1e})$、$P_2=[P_{2s},P_{2e})$是时间期间,len 是时间跨度。

(1) 时间期间与时间跨度加法。时间期间 P+时间跨度 len=$[P_s+\text{len},P_e+\text{len})$。

此时运算语义为:在时间轴上的时间期间 P 向右平移了 len 个时间单位。

运算满足交换律:P+len=len+P。

(2) 时间期间与时间跨度减法。时间期间 P-时间跨度 len=$[P_s-\text{len},P_e-\text{len})$。

此时运算语义为:时间轴上的时间期间 P 向左平移了 len 个时间单位。

(3) 时间期间与时间期间减法。当 $P_{2s} \leqslant P_{1e}$时,时间期间 P_1-时间期间 P_2= $P_{1e}-P_{2s}$。

当 $P_{2e} \leqslant P_{1s}$时,时间期间 P_1-时间期间 P_2=$P_{1s}-P_{2e}$。

此时运算语义为:时间轴上两个时间期间 P_1 和 P_2 之间的距离。

3) 时间跨度间运算

两个时间跨度之间可以定义相应的加、减、乘和除法运算,但在相除需要将商的小数部分截断而仍然为整数以表示一个时间跨度。作为整型数据,时间跨度和整数也可以进行相应运算。设 len 为时间跨度,i 是一个整数,则可以定义 len 与 i 的乘法与除法运算如下:

(1) len×i=i×len,运算结果仍然是一个时间跨度。

(2) len÷i,运算结果仍然是一个时间跨度。

4) 时间元素聚集函数

设 p_1、p_2 是时间点,P=[P_s,P_e]是时间期间,定义如下聚集运算。

(1) 两时间点最小函数 min(p_1,p_2)。

(2) 两时间点最大函数 max(p_1,p_2)。

(3) 求时间期间始点函数 begin(P)=P_s。

(4) 时间期间终点函数 end(P)=P_e。

(5) 时间期间跨度函数 length(P)=P_e−P_s。

2. 时间关系运算

时间关系运算是使用时间谓词表达式的必备前提,主要用于判断和处理时间期间相互之间的位置拓扑关系。现有的时间关系运算主要是采用 Allen 提出的 13 种基于时间期间的关系演算,如表 6-1 所示。其中,P_1、P_2 分别表示时间期间。

表 6-1 Allen 提出的 13 种基于时间期间的关系演算

时间期间关系	基本语义
Before(P_1,P_2)	P_1 比 P_2 早开始,同时 P_1 与 P_2 之间没有相交
After(P_1,P_2)	P_1 比 P_2 晚开始,同时 P_1 与 P_2 之间没有相交
During(P_1,P_2)	P_1 比 P_2 晚开始,且早结束,即在时间轴上 P_1 的区间范围被包含在 P_2 的区间范围内
Contains(P_1,P_2)	P_1 比 P_2 早开始,且晚结束,即在时间轴上 P_1 的区间范围包括了 P_2 的区间范围
Overlaps(P_1,P_2)	P_1 比 P_2 早开始,且早结束,且两个区间在时间轴上有相交
Overlapped-by(P_1,P_2)	P_1 比 P_2 晚开始,且晚结束,且两个区间在时间轴上有相交
Meets(P_1,P_2)	P_1 比 P_2 早开始,且 P_1 与 P_2 之间没有其他时态区间,即 P_2 开始于 P_1 的结束点
Met-by(P_1,P_2)	P_1 比 P_2 晚开始,且 P_1 与 P_2 之间没有其他时态区间,即 P_1 开始于 P_2 的结束点
Starts(P_1,P_2)	P_1 和 P_2 有共同的起始点,但 P_1 比 P_2 先结束
Started-by(P_1,P_2)	P_1 和 P_2 有共同的起始点,但 P_2 比 P_1 先结束
Finishes(P_1,P_2)	P_1 和 P_2 有共同的结束点,但 P_1 比 P_2 晚开始
Finished-by(P_1,P_2)	P_1 和 P_2 有共同的结束点,但 P_2 比 P_1 晚开始
Equals(P_1,P_2)	P_1 和 P_2 有共同的时间区间,即 P_1 与 P_2 在时间轴上重合

在上述 13 种时间期间关系中,有 6 对关系可以互相转换,即 Before(P_1,P_2)=After(P_2,P_1)、During(P_1,P_2)=Contains(P_2,P_1)、Overlaps(P_1,P_2)=Overlapped-by(P_2,P_1)、Meets(P_1,P_2)=Met-by(P_2,P_1)、Starts(P_1,P_2)=Started-by(P_2,P_1)、Finishes(P_1,P_2)=Finished-by(P_2,P_1)。另外还有 Equals(P_1,P_2)=Equals(P_2,P_1),因此实际上只有 7 种关系演算独立。

6.1.4 时间维度与时态数据库

客观事物都处在一定时间环境当中,并随时间演进发展变化。数据信息是客观事物的反映,因而也随时间而变化。随时间变化的数据可看作是时态数据(temporal data)。时态数据中的时间数据信息可以隐式表示,如 RDB 和 ODB 等中的数据;也可以根据需要或必

须显示表示,如本章所要讨论的 TDB 中的时态数据。时态数据通常通过时间标签(timestamps)描述其数据相应时间特征。描述现实世界的带有时间属性的数据库系统,尤其是以时态数据管理为主要特征的系统通常称为时态数据库系统(Temporal DataBase System,TDBS)。TDBS 的典型应用如有金融方面的会计和银行数据系统;档案管理方面的人事档案和医疗记录数据系统;行程安排中的飞机、火车和酒店等的数据系统;科学应用方面气候和地理数据系统等。表 6-2 表示一个单位员工的职称和工资随时间变化的情况。其中,员工的姓名具有相对稳定性,时变属性职称和工资记录是分段的。在这里,用表格方式记录的该单位员工工资和职称的数据就是时态关系数据,这些时态数据反映该单位员工职称和工资信息就是时态信息。

表 6-2　员工的职称和工资随时间变化的情况

Name	Title	Salary
Rose	Lecture.[2005,2010) Associatt-prof.[2011,2014)	7000,[2005,2010) 9500,[2011,2014)
White	Lecture.[2012,2015)	5500,[2012,2015)
Peter	Prof.[2011,2013)	12000,[2011,2013)

在时态数据库系统中,通常不是考虑单个时间元素而是若干相关时间元素的一个集合,同时还需要根据实际应用情形研究时间元素集合的特定语义即数据的时间维度(time demension)。在实际应用中,时间维度主要有用户定义时间维度、有效时间维度和事务时间维度 3 种基本情况。

1. 数据的时间维度

物理实体在空间的 3 个维度实际上可以看作是观察该实体的 3 个不同角度。因此,在分析考虑一般事物时,人们也常常将考察的角度称为维度。为了在计算机中实现时间的有效管理,需要将数据所涉及时间进行各种角度的解构分析,针对不同情形进行处理整合。在时态数据管理框架内,所涉及时间通常分为用户定义时间、有效时间和事务时间 3 个维度。

1) 用户定义时间

用户自定义时间(user-defined time)是指用户根据自身需要或理解而定义的时间。通常取值为时间点或用户定义数据类型,相应语义由用户应用本身予以解释。

DBMS 将用户定义时间作为普通属性与其他属性同等对待处理,一般都将其归结为普通字符串操作。系统不对其进行任何特殊处理,也不提供专门语言支持。

用户定义时间的提供和更新由用户自身完成。用户定义时间值完全依赖于实际应用,由用户和系统以常规方式存取。例如,为了存储"生日"信息,可以根据需要定义一个"生日"数据类型的属性,如相应元组中对应属性值为"1998-10-21",此时"生日"就是一个用户定义时间。

由于允许在原有系统数据类型的基础上建立用户自身所需数据类型,RDBS 支持用户定义时间数据类型。在创建或更新数据时,用户定义时间的数据类型和其他数据类型一样被用户使用。

2) 有效时间

有效时间(valid time)是指一个对象(事件)在现实世界中发生并保持的那段时间,或者

该对象在现实世界中为真的时间。

有效时间可以是单一的时间点、单一的时间期间或者是时间点和时间期间有限集合及混成集合,也可以是整个时间域,这是由于一条记录属性取值可能在任意时间点、时间期间内为真。有效时间的创建和更新由用户自身完成。但与用户自定义时间不同的是:相应查询语句被检测到存在有效时间语义时,需要通过DBMS进行解释。

有效时间有如下主要特点。

(1) 有效时间值的含义依赖于具体应用,取值是否有效由具体应用场合而定,即涉及(时态)数据约束问题。

(2) 有效时间一般具有过去时间、现在时间和未来时间的基本语义。

有效时间对事物的描述简洁直观、容易理解。表6-3是具有有效时间关系表的实例。

表6-3 具有有效时间关系表的实例

Name	Title	VTs	VTe
Raul	Lecture	2008-07-01	2011-09-30
Raul	Associate-prof.	2011-10-01	2013-05-31
Raul	Prof.	2013-06-01	Now

由表6-3可知,Raul身份变动历史,通过增加起始有效时间VTs(starting valid time)和增加终止有效时间VTe(ending valid time)两个属性记录数据有效时间。但增加这两个属性并不意味RDBMS就自动可以转换为TDBMS,这是因为作为一个TDBMS,需要支持时态数据定义语言(TDDL)、时态数据操作语言(TDML)、时态查询语言(temporal query language)和时态约束(temporal constraints),常规RDBMS并不具备复杂深入的功能。

3) 事务时间

事务时间(transaction time)是指对给定数据库对象进行数据操作,如插入、删除或修改的时间,即一个事实进入并存储于数据库当中的时间。

事务时间记录对数据库操作的各种历史情形,对应于现有事务或现有数据库状态变迁的历史。例如,数据录入数据库的时间,对其进行查询的时间和对其进行删除或修改的时间等。

事务时间对应于现有事务或现有数据库的状态变迁历史,独立于相应实际应用,用户不能对事务时间进行任何处理。数据库中数据录入、修改和删除的时间由系统时钟决定,每次更新后相应时间数据不可再予以改变,因此,事务时间也称为系统时间(system time)。

处理事务时间的方法是存储所有数据库的状态,即处理一个事务之后就存储一种数据库状态。任何对数据的更新只能针对最后一个状态进行,但可查询任意一个状态。

事务时间具有下述主要特点。

(1) 事务时间取值由系统时钟给出,独立于应用,不允许用户对事务时间进行任何修改。

(2) 最新记录的事务时间不能晚于系统当前时间,它反映数据库实际操作的时间,不能表示未来时间语义。

需要注意有效时间和事务时间的区别。

(1) 有效时间标识一个数据对象在现实的世界中发生并保持的时间,或者说使得该数据对象在现实世界中为真的那段时间,本质上对应于实际应用的需要或现实世界的变化。

(2) 事务时间记录逻辑上被存放在数据库的时间，反映记录被查询、删除和修改的时间。

(3) 事务时间与有效时间相互独立，彼此正交，从不同方面描述了数据和数据库随时间演进的变化过程。

2. 时态数据库

常规数据库通常只保留一个企业或单位的当前状态。由于客观事物总是要随着时间发展变化，当前状态会被进一步发展的状态所取代。常规数据库作为时间维度上的一个快照，在一般意义都不保存历史数据。在实际应用中，不仅当前信息，在某些情况下过去已有信息甚至将来预测信息都非常重要。例如，人事管理系统中的个人履历，财务系统中的往来账目，地理信息系统中的各种地质变化记录，医疗系统中的诊断记录和病人病历等。这就要求数据库能够在不同的时间维度上充分展开，有效管理各种与时间演进有关的数据信息。常规数据库系统如 RDBS 和 ODBS 也可以从事时间数据管理方面工作。例如，在 RDBS 中，可以将元组的生命周期用一个属性表示，即将时间作为普通属性进行操作。既然时间作为了普通属性，则当进行关系操作如查询（选择、投影和连接）与更新（插入、删除和修改），就不可避免需要对时间属性进行关系运算（关系代数和关系演算）。但时间元素的运算有其自身特点，不能简单纳入常规属性范围，如当一个元组的生命周期需要用时间元素表示时就得到时间属性值的非原子表示，常规 RDBMS 处理起这类问题相当笨拙。实践表明，如果直接使用 RDBS 等管理时态数据，通常会出现时间描述粗放、数据管理困难和查询结果不符合常理直观等问题。特别是有关时间运算相当复杂，必须借助于大量的应用程序，脱离了数据库管理数据的本意，增加系统复杂性，也加重用户负担。

【例 6-1】 设有如表 6-4 所示关系表，各个元组有效时间（VTs，VTe）分别表示为时态属性 VTs 和 VTe。执行下述 SQL 语句：

```
SELECT Salary, VTs,VTe
FROM TR
```

常规关系数据查询结果如表 6-5 所示。

表 6-4 具有时态属性的关系表 TR

No	Name	Salary	Title	Position	VTs	VTe
0199804426	Raul	4500	Lecture	Null	2008-07	2011-09
0199804426	Raul	6500	Associate-Prof.	Assistant-President	2011-10	2013-05
0199804426	Raul	9200	Professor	Vice-President	2013-06	2014-08
0199804426	Raul	9200	Professor	President	2014-08	2016-08

表 6-5 常规查询结果

Name	Salary	VTs	VTe
Raul	4500	2008-07	2011-09
Raul	6500	2011-10	2013-05
Raul	9200	2013-06	2014-08
Raul	9200	2014-08	2016-08

在表 6-5 所示的查询结果中，第 3 元组和第 4 元组非时态部分完全相同，而时态部分“可连接”，因此，输出如表 6-6 所示结果就会更加自然直观，同时也减少了数据冗余。但在常规 RDBS 中却必须以表 6-6 形式输出。如果希望表 6-6 形式，则需通过额外应用程序。事实上，这只是进行常规投影操作情形，在选择和连接操作中也会出现类似问题。

表 6-6　符合直观的查询结果

Name	Salary	VTs	VTe
Raul	4500	2008-07	2011-09
Raul	6500	2011-10	2013-05
Raul	9200	2013-06	2016-08

一般而言，DBMS 作用就是将各种基本功能装配于系统之中，而不是通过各种应用程序来完成相应数据管理任务，因此，研制和开发 TDBS 就成为一种基本需求和管理时态数据的必然。

按数据时间维度不同，时态数据库分为快照数据库、回滚数据库、历史数据库和双时态数据库 4 种类型。但通常的时态数据库(temporal database)主要指后 3 种类型。

1）快照数据库

快照数据库(snapshot database)是指以特定时刻瞬间的快照进行数据库建模。

快照数据库无法表示属性与时间的关系，不具备维护状态变迁能力，只能进行当前数据库状态的查询和更新，难以进行对历史数据查询，且随着时间演进，更改的历史数据将会丢失，也不能进行含有时间因素的推理。

例如，快照数据库难以完成下述查询：

“Raul 何时开始担任讲师？(如果他现在是副教授)”(历史查询)

“2016 年 9 月 18 日的记录中，Raul 的职务是什么?”(历史查询)

“在过去 3 年里，中山大学有多少人从副教授提升为教授?”(趋势查询)

“明年，Raul 是否能够晋升为博士生导师?”(未来查询)

“Raul 上个月工资由 9200 元提升为 12000 元。”(记录更新)

此外，快照数据库中的“快照”和 RDB 中的“快照”有所不同。

(1) RDB 中的“快照”：为了处理需要(如年底结账的需要)对某个时刻(如 12 月 31 日 23 时 59 分 59 秒)数据库中数据进行独立备份。

(2) 快照数据库中的“快照”：系统只存储管理数据库当前一个快照状态，而“快照”状态随着时间不断改变。

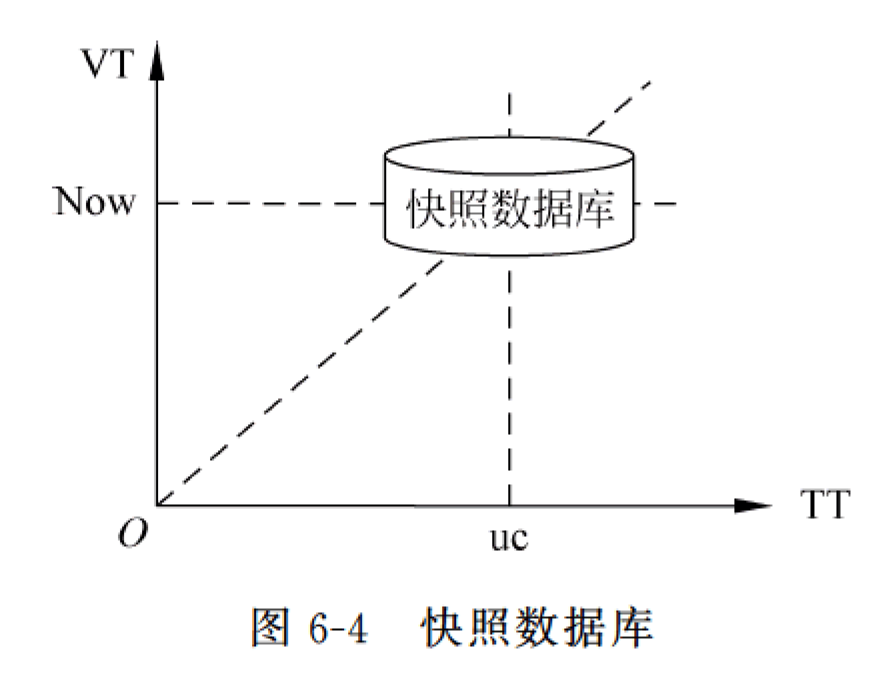


图 6-4　快照数据库

快照数据库支持用户定义时间，但不需要区分数据的事务时间和有效时间。如图 6-4 所示，快照数据库可看作是沿着对角线方向前进。

快照数据库是数据库当前的一个感知状态，且“快照”状态随时间不断改变。快照数据库并非真正意义下的时态数据库，只反映数据当前状态，时间推移将导致快照数据库状态不断改变，新状态将覆盖旧的状态。

2）回滚数据库

回滚数据库(rollback database)是指支持数据事务时间的时态数据库。

回滚数据库中数据按照事务时间编址，保存过去每次事务提交和状态演变之前的状态。从 RDB 进行事务时间的时态扩充角度考虑，可以将回滚数据库中的数据文件看作是由三维回滚关系组成的，即在传统属性维和元组维基础上增加事务时间维。因此，回滚数据库中的数据表可以看作一个按时间索引的数据库序列，每个时间点都对应一个二维快照数据库。回滚数据库记录数据库的自身变化，实现方式是沿着事务时间轴记录数据状态，按照事务时间排序保留所有状态的演变历史，可被看作是只能进行数据追加的数据库，但不能用来记录数据库的未来状态。执行一次更新语句都将产生一个新的数据库状态。新状态不会覆盖旧状态，这是因为系统是通过将一个元组的事务结束时间设为执行语句的当时时间而实现数据的逻辑删除，因此没有元组会被物理删除，元组的事务时间期间可以看作为该元组在快照数据库中存在的历史。

作为一个由事务时间索引的数据库序列，回滚数据库在前一个事务时间内提交的数据，即使在下一个事务时间没有数据改变或者改变很小时，也需将所有数据重新输入及存储，带来较大的数据冗余，这在只进行较小数量数据改变时更为突出。在回滚数据库中，过去数据只能查询而不可修改。如果发现已有数据存在错误，当此时事务已经提交后就不可予以变更，只能是等待下次系统的事务时间时才进行新的改动。但改动的只是提交前的数据，即最近一个事务时间点的数据，此前的状态并不能再改变。

3）历史数据库

快照数据库考察特定时刻下现实世界的一个状态，反映了某一个瞬间的情况。由如表 6-7 所示的快照数据库中可知道相应教师的一些基本信息。

表 6-7　快照数据关系

No.	Name	Birthday	Title
0199804426	Raul	1969-06-06	Lectuer
0199804427	White	1966-07-08	Prof.
0199804428	Bush	1963-08-16	Prof.

对于“Raul 在两年前是否为副教授”这类历史查询，除非对快照数据关系结构进行特殊的应用程序处理，否则将难以得到所需结果。为解决此类问题，就需引入历史数据库。

历史数据库(historical database)是指存储数据有效时间的时态数据库，存储现实世界在有效时间点处或有效时间期间内的事件和状态变化。

在表 6-7 所示的快照数据关系中添加有效时间支持，并将其中部分改写为如表 6-8 所示，就成为历史数据库中时态关系数据表。

表 6-8　添加有效时间的时态数据关系

No.	Name	Birthday	Title	VTs	VTe
0199804426	Raul	1967-09	Lecture	2008-07	2011-09
0199804426	Raul	1967-09	Associate-Prof.	2011-10	2013-05
0199804426	Raul	1967-09	Professor	2013-06	2016-08

续表

No	Name	Birthday	Title	VTs	VTe
0199804427	White	1969-02	Associate -Prof.	2012-08	2015-08
0199804427	White	1969-02	Professor	2015-09	2016-09
0199804428	Bush	1973-10	Lecture	2013-08	2016-08
0199804428	Bush	1973-10	Associate-Prof.	2016-09	2017-09

若当前为2016年，对于上述“Raul两年前是否为副教授?”时态查询，则可知两年前，即2014年Raul已经不是副教授而是教授了。

相对于快照数据库存储数据库当前状态信息，历史数据库通过支持有效时间而增加系统所包含的历史数据信息量，方便人们对历史数据信息的查询和处理。

回滚数据库记录数据库状态的事务变迁，历史数据库记录了数据对象状态的现实变迁。

从RDB进行有效时间的时态扩充角度来看，历史数据文件是由具有属性维、元组维和有效时间维构成的三维关系表。其中，有效时间维度反映了相应由属性维和元组维构成二维关系数据的“生命周期”，使得相关数据信息更为丰富，对客观事实的描述表现更为深入。从原理探讨和技术实现的角度来说，历史数据库是时态数据库的研究主题和基础。

4）双时态数据库

回滚数据库是依照事务时间索引的数据库快照系列，历史数据库通过有效时间存储事物的生命周期，可否建立一种能够同时处理事务时间和有效时间的时态数据库呢?

双时态数据库(bitemporal database)就是同时支持数据事务时间和有效时间的时态数据库。

从RDB进行事务时间和有效时间的时态扩充角度来看，双时态数据库由四维时态关系表系列组成。其中，两个是常规属性维度和元组维度；另外两个是事务时间维度和有效时间维度。由此可知，一个双时态关系可以看成是一个按照事务时间索引的历史关系的序列。双时态数据架构如图6-5所示。

在时态关系的数据操作中，其回滚操作即是选取一个特定的历史关系，其查询操作即是对该历史关系进行查询；同时每一个更新的事务操作即是创建一个新的历史关系表。

双时态关系的一种实现方法是组合回滚数据库和历史数据库成为新的数据库。

一个常规关系元组在历史数据库中的四个有效时间片段组合如图6-6所示，图中是在原有回滚关系的三维结构之上添加第四个维度——有效时间维度，而使得相应数据库具有了四维结构。

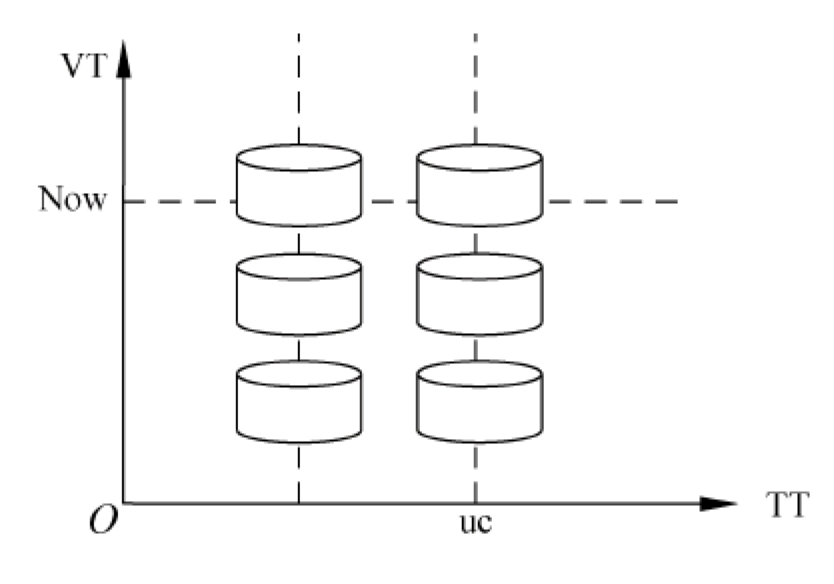


图6-5　双时态数据架构

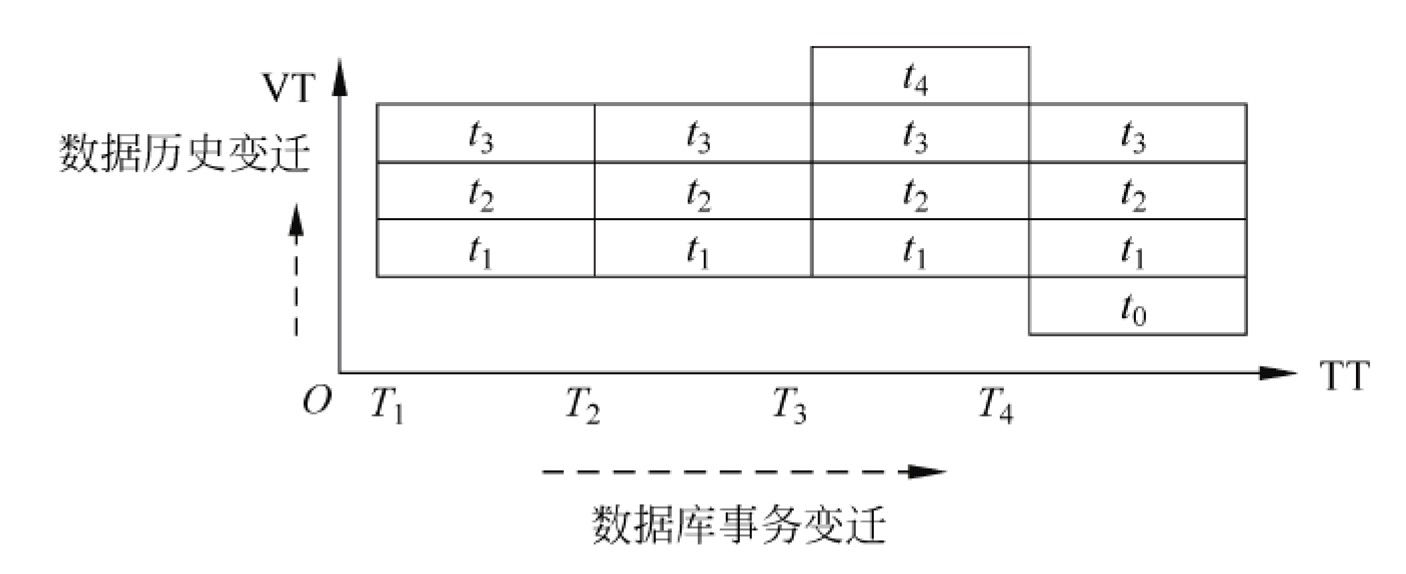


图6-6　双时态数据关系的两个时间维度

在事务维度中截取事务时间点就可找到相应元组的有效时间段，不同事务时间点对应不同有效时间段。同时，相同的有效时间段也可对应不同的事务时间点，如图 6-6 中事务时间点 T_1 和 T_2 对应同一有效时间段。

实际上，在事务时间轴上取不同时间点就产生不同的历史数据库，如图 6-6 中 4 个事务时间点 T_1、T_2、T_3、T_4 就对应 4 个历史数据库，可以将双时态数据库看作是基于事务时间点索引的历史数据库系列。由于是对相应事务时间点的“截取”，因此，这 4 个数据库是分别是 4 个事务时间的快照；又因为每个数据库里面的记录是历史数据库属性的，记载的是现实元组的真实变化的时间，而非数据库事务状态变化的时间，所以就是前述的历史数据记录，可以在其中进行查询与更新的常规数据操作。

双时态数据库中可在当前时间点对以前的事务时间 T_1 时的该元组属性或有效时间进行改动。例如，可以在 T_4 时间对 T_1 时的历史快照数据库进行修改，通过改变有效时间期间 t_1、t_2 和 t_3 为 t_1 和 t_3，使得在 T_1 时的快照历史数据库中的元组属性（时间属性）得到了改变。但原先事务时间不能改动，只是增加了一个新的记录，该记录的事务时间是 T_4，记录内容是把原来的有效时间进行了改变。

6.2　历史关系数据模型

历史关系数据模型（History Relational Data Model，HRDM）由 James Clifford 于 1987 年提出，这是一个基于有效时间的时态数据模型。HRDM 基本思想是为常规关系表中的基本技术单元属性（值）、元组和关系表赋予满足相应约束条件的有效时间标签。

6.2.1　HRDM 概述

给定时间域为 T，其中 T 由时间期间或时间点组成。在 TDB 中，从一般性角度考虑，通常将所涉及的系统时间域看作 T 的幂集 2^T。

1. 数据对象有效时间

数据对象的有效时间实际上可看作其生命周期，而生命周期通常是一个时间元素而属于系统时间域 2^T，它表示所考虑数据对象在现实世界为“真”的时间。RDB 基本数据对象是属性（值）、元组和关系表，因此需要讨论这些数据对象的有效时间问题。

（1）属性值有效时间：属性值的生命周期。例如，员工工资随着时间变化的情况如表 6-9 所示。工号 ID 为“2010519795012”、员工工资为 3500 元，有效时间是 2010 年到 2013 年，即“工资为 3500 元的生命周期为 2010—2013 年”；类似，“工资为 4200 元的生命周期为 2014—2015 年”和“工资为 5900 元的生命周期为 2016 年到 Now”。作为属性时间变化表述的时间期间[2010，2013)，[2013，2015)、[2016，Now)等就是相应工资属性值生命周期即有效时间。另外，此时属性值“2010519795012”没有显式赋予有效时间，实际上由于“2010519795012”是键值，其生命周期应该由整个关系表生命周期所确定，因此可以将关系表中所有“键值”的有效时间都看作“常值”即所在关系表的有效时间。

（2）元组有效时间：整个元组自身的生命周期。如表 6-9 所示，ID 为“2010519795012”元组有效时间为[2010，Now)。

（3）关系表有效时间：可以理解为组成其所有元组的有效时间的并集。

表 6-9　员工工资随着时间变化的情况

ID(key)	Salary	Valid time
2010519795012	3500[2010,2013)	[2010,Now)
	4000[2014,2015)	
	6000[2016,Now)	
……	……	……

整合了有效时间标签的属性(值)、元组和关系通常分别称为时态属性、时态元组和时态关系。

从直观上考虑,属性值有效时间不应超过所在元组有效时间,元组有效时间不应超过所在关系表生命周期,由此可得,时态属性、时态元组和时态关系表之间的时态语义约束条件为:

属性有效时间⊆元组有效时间⊆关系有效时间

对于上述属性和元组有效时间表述而言,由于有效时间标签一般是时间元素,因此相应关系表可能是非 1NF,并且属性值和元组相应有效时间标签可能互不相同。如果设定相应关系表满足 1NF,就需要应用经典关系模式规范中的非 1NF 解构为 1NF 方法,由于有前述时态约束,解构后关系模式中属性有效和元祖有效时间就会彼此相同。

2. HRDM

历史关系数据模型(HRDM)就是将上述对属性值和元组有效时间赋值的严格化和形式化。HRDM 建立在时间域 T 与相关属性域的映射基础之上。

1) 有效时间赋值映射

HRDM 的基本点是为每个属性(值)、元组和关系表赋予适当的有效时间标签,使其成为时态属性、时态元组和时态关系。从形式化角度考虑,这实际上就是建立起属性集合、元组集合和关系集合到时间域上的相关映射。

设有关系属性集合$\{A_1,A_2,\cdots,A_m\}$,对应属性值域集为$\{D_1,D_2,\cdots,D_m\}$,T 为时间期间(时间点)集合即 T 为时间域,2^T 为 T 的幂集。

(1) 属性值有效时间赋值映射 VTD_i:对于每个属性值域 D_i,定义有效时间赋值映射 VTD_i:$D_i \to 2^T (1\leqslant i\leqslant m)$。

(2) 属性值有效时间常值映射 CT:对于一个属性值域或属性域集合 D_0,定义由属性值域 D_0 到 T 的常值映射 $CT(D_0)$:$D_0 \to t_0$,t_0 为时间域 T 中一给定值。

(3) 元组有效时间赋值映射 VTT_j:对于关系表 R 中所有元组构成集合$\{T_1,T_2,\cdots,T_n\}$,定义元组有效时间赋值映射 VTT_j:$T_j \to 2^T (1\leqslant j\leqslant n)$。

2) HRDM 基本概念

基于 HRDM 模型的时态关系模式 TR 为满足时间约束 TC 的五元组:

$TR=\langle A(TR),K(TR),\{VTD_i\}_{1\leqslant i\leqslant m},\{VTT\}_{1\leqslant j\leqslant n},CT(K(TR))\rangle$

(1) A(TR):TR 所有属性集合$\{A_1,A_2,\cdots,A_n\}$。

(2) K(TR):TR 主键属性集合。

(3) $\{VTD_i\}_{1\leqslant i\leqslant m}$:属性域到时间域的映射集合。

(4) $\{VTT\}_{1\leqslant j\leqslant n}$:元组集合到时间域的映射集合。

(5) CT(K(TR):键值集合到时间域的常值映射。

时态约束条件 TC 如下。

设 T_0 是时态关系 TR 中的任意元组，a_k 是元组 $T_0=<a_1,a_2,\cdots,a_s>$中属性值，则：

$$VT(a_k)\subseteq VT(T_0)\subseteq VT(TR)(1\leqslant k\leqslant s)$$

3）HRDM 属性值非 1NF 性质

由于 HRDM 是常规关系的时态扩充，因此元组数据的语义标识还是主键。主键的基本性质是相应元组的唯一标识性，因此和常规关系情形类似。在 HRDM 数据模式中，时态关系 TR 中不允许出现主键值相同的两个元组，即

$$\forall Ttup_1、Ttup_2\in TR, Key(Ttup_1)\neq Key(Ttup_2)$$

由于历史数据库需要记录同一数据对象在不同时间的状态和性质，这也包括同一属性在不同时间范围内中的不同取值，因此 HRDM 关系中确定主键值所在元组中的属性取值不会是完全满足 1NF，这也就导致了 HRDM 中的属性取值通常具有非 1NF 性质。

基于 HRDM 时态关系，TR 可以表示为时态关系元组 Ttup 的集合，其中 $Ttup=<\{a_k\}_{1\leqslant k\leqslant s}, VT(Ttup)>$，$a_k$ 是元组中第 k 个属性的时态取值。由上述分析可知，a_k 一般应为如下的非 1NF 形式，即

$$a_k=<(v_1(a_k),\ VT(v_1(a_k))),(v_2(a_k),\ VT(v_2(a_k))),\cdots,(v_p(a_k),VT(v_p(a_k)))>$$

即 a_k 是相应数据值 $v_r(a_k)$和有效时间标签 $VT(v_r(a_k))$构成二元组的一个有序组($1\leqslant r\leqslant p$)。

例如，在表 6-9 中，ID(主键)为 2010519795012 的元组中 Salary 属性值可表示为：

$$a_k=<(3500,\ [2010,2013)),(4000,\ [2014,2015)),(6000,\ [2016,Now))$$

HRDM 模型简洁直观，容易掌握，基本要点是在常规 RDB 上添加上属性和元组的生命周期从而对经典 RDB 进行兼容扩展。作为时态数据库，基于 HRDM 建立的是历史数据库只能管理对象的历史，不能管理数据库本身插删改的历史即事务时间。

在 HRDM 中，当有效时间退缩为一点$[t,t)$时，得到在 t 时刻的快照数据库，此时 HRDM 所有运算都退化为传统关系运算，而传统 RDB 当前状态可视为在时间期间[Now,Now)的特例。

RDM 中许多基本运算的规则都可平移到 HRDM，如选择运算的交换律、结合律及与其他运算的分配律等。

6.2.2 HRDM 数据操作

在 RDB 引入有效时间后，关系代数操作将具有一些新的特点和变化。下面介绍 HRDM 中时态关系操作，即时态数据更新和时态数据查询。

1. 时态数据更新

关系数据更新的基本操作是数据删除和数据插入。对于基于 HRDM 的时态关系而言，由于不允许存在主键相同的两个元组，按照元组的标识唯一性就可以进行与常规关系类型的数据删除操作；但由不同元组主键值必须不同却导致了数据插入时需要考虑的新的情形。实际上，时态关系中插入新元组时有如下两种不同的情形。

(1) 插入元组主键与表中任何元组主键都不相同，此时按照常规关系方法实现插入操作。

(2) 插入元组主键和某个已有元组主键相同，此时需要完成下述操作。

① 当插入元组和该已有元组完全相同即两者所有属性值和属性值有效时间都相同时，类似于常规 RDB，两元组只保留其一。

② 对于存在属性值相同而属性有效时间不同的情形，相应有效时间存在包含关系，取

“小”的有效时间作为相应属性值新的有效时间予以保留。

③ 对于属性值和相应有效时间都不相同的情形，将插入元组和该已有元组对应属性值进行归并。

【例 6-2】 设有如表 6-10 所示时态关系表。

表 6-10 时态关系表

Name	Salary	Valid time
Peter	3500[2010,2011)	[2010,Now)
	4000[2012,2013)	
	6000[2014,Now)	
Bush	3300[2008,2010)	[2008,Now)
	4500[2011,2013)	
	6000[2014,Now)	

插入如表 6-11 和表 6-12 所示两个时态关系元组。

表 6-11 新插入 Peter 元组

Peter	4000[2012,2013)	[2012,Now)
	6000[2014,2015)	
	7500[2016,Now)	

表 6-12 新插入 Bush 元组

Bush	4500[2011,2013)	[2011,Now)
	6000[2014,2015)	
	7600[2016,Now)	

对于新插入 Peter 元组中时态属性值“4000[2012,2013)”而言，其与表 6-10 中原有元组中相应时态属性值“4000[2012,2013)”相同，保留其一。

新插入时态属性值“6000[2014,2015)”与原有时态属性值“6000[2014,Now)”中只有时间标签不同，而[2014,2015)⊆[2014,Now)，所以予以保留的新时态属性值为“6000[2014,2015)”。

另外，新插入时态属性值“7500[2016,Now)”直接归并到相应属性列。

对于新插入 Bush 元组也类似处理。插入表 6-11 所示元组后的时态关系表如表 6-13 所示。

表 6-13 插入更新后的关系表

Name	Salary	Valid time
Peter	3500[2010,2011)	[2010,Now)
	4000[2012,2013)	
	6000[2014,2015)	
	7500[2016,Now)	
Bush	3300[2008,2010)	[2008,Now)
	4500[2011,2013)	
	6000[2014,2015)	
	7600[2016,Now)	

2. 时态数据查询

基于 HRDM 数据查询也分为投影、选择和连接 3 种情形。

1）时态投影

时态关系数据的投影运算与 RDB 中类似，只是投影运算后需根据情况将属性值或元组归并，并将相应有效时间进行调整。由于 HRDM 时态关系允许属性值取非 1NF，由此时态属性值和时态元组的归并比较复杂，通常需要考虑到实际应用。

【例 6-3】 设有如表 6-14 所示的某单位的历史数据时态关系表。

表 6-14　历史数据时态关系表

<table>
<tr><th>Name</th><th>Title</th><th>Salary</th><th>Valid time</th></tr>
<tr><td rowspan="4">Peter</td><td>Lecture[2010,2011)</td><td>3500[2010,2011)</td><td rowspan="4">[2010,Now)</td></tr>
<tr><td rowspan="2">Associate-Prof. [2012,2015)</td><td>4000[2012,2013)</td></tr>
<tr><td>6000[2014,2015)</td></tr>
<tr><td>Prof. [2016,Now)</td><td>7500[2016,Now)</td></tr>
<tr><td rowspan="4">Bush</td><td>Lecture[2008,2013)</td><td>3300[2008,2010)</td><td rowspan="4">[2008,Now)</td></tr>
<tr><td rowspan="2">Associate-Prof. [2014,2015)</td><td>4500[2011,2013)</td></tr>
<tr><td>6000[2014,2015)</td></tr>
<tr><td>Prof. [2016,Now)</td><td>7600[2016,Now)</td></tr>
</table>

现在需要了解一个单位中 Lecture、Associate-Prof. 和 Prof. 职称设置信息，如“在怎样时间期间内有讲师”和“在怎样的时间期间内有高级职称（副教授和教授）”等。此时查询可以分为两个步骤。

步骤 1：将其进行 Title 属性的常规投影操作，如表 6-15 所示。

表 6-15　在 Title 属性的常规投影

<table>
<tr><th>Title</th><th>Valid time</th></tr>
<tr><td>Lecture[2010,2011)</td><td rowspan="3">[2010,Now)</td></tr>
<tr><td>Associate-Prof. [2012,2015)</td></tr>
<tr><td>Prof. [2016,Now)</td></tr>
<tr><td>Lecture[2008,2013)</td><td rowspan="3">[2008,Now)</td></tr>
<tr><td>Associate-Prof. [2014,2015)</td></tr>
<tr><td>Prof. [2016,Now)</td></tr>
</table>

步骤 2：将上述结果调整归并，如表 6-16 所示。

表 6-16　进行归并调整

<table>
<tr><th>Title</th><th>Valid time</th></tr>
<tr><td>Lecture[2008,2013)</td><td>[2008,2013)</td></tr>
<tr><td>Associate-Prof. [2012,2015)</td><td rowspan="2">[2012,Now)</td></tr>
<tr><td>Prof. [2016,Now)</td></tr>
</table>

2）时态选择

HRDM 在常规关系数据模式基础上增加了数据有效时间，时态关系选择操作除对非时

态属性进行通常选择运算外还要进行相应有效时间的选择操作。选择运算基础是相应的谓词选择表达式,为此需要引入时态选择谓词表达式。

时态选择谓词表达式是由逻辑运算符(∨、∧、¬)、关系运算符(>,≥,<,≤,=,≠)、集合运算符(∪、∩、—)和 Allen 时态关系运算符组成的谓词表达式。

时态选择操作就是在时态关系表中选择出满足给定时态选择谓词表达式所有元组。按照选择结果中有效时间标签是否发生改变,基于 HRDM 时态关系选择操作可以分为两种情形。

(1) Select_IF 选择。类似常规关系表选择操作,选择出元组的有效时间与关系表中相应元组相同。

【例 6-4】 设有如表 6-17 所示时态关系表。

表 6-17 时态关系表

Name	Salary	Valid time
Peter	3500[2009,2010)	[2009,Now)
	4000[2011,2012)	
	6000[2013,2015)	
	7500[2016,Now)	

对上述时态关系进行操作 $\text{Select_IF}_{\text{Salary}=6000\wedge[2014,2015)}$,选择结果如表 6-18 所示。

表 6-18 Select_IF 选择操作

Name	Salary	Valid time
Peter	6000[2013,2015)	[2009,Now)

由此可知,Select_IF 选择只排除不合格的对象,不缩小对象在有效时间维上的尺寸,可以看作只是数据事务维度上的筛选。

(2) Select_When 选择。先进行 Select_IF 选择,将时态选择谓词表达式中有效时间作为选择结果的有效时间标签。

对于表 6-17 所示时态关系执行选择 $\text{Select_When}_{\text{Salary}=6000\wedge[2014,2015)}$,选择结果如表 6-19 所示。

表 6-19 Selest_When 选择操作

Name	Salary	Valid time
Peter	6000[2014,2015)	[2014,2015)

由此可知,Select_When 不但排除了不合格的元组,而且还在时间维上排除了不合格的部分,可以看作是关于数据事务和时间维度上的双重筛选运算。

3) 时态连接

作为常规关系连接运算拓展,HRDM 时态关系上时态连接操作还需考虑有效时间因素。

两个 HRDM 时态关系进行时态连接操作的步骤如下。

(1) 选择出满足常规连接条件的元组。

（2）对于这样常规可连接的两个元组，考察元组有效时间是否有交。如果相交，则连接后元组有效时间就是这两有效时间的交集，否则不可进行时态连接。

（3）检查连接后属性值数据的有效时间是否包含在连接元组有效时间之内，如果是，则保留。否则，当有交时，以交作为相应有效时间；无交时，删除该属性值数据。

【例 6-5】 设有如表 6-20 和表 6-21 所示 HRDM 时态关系 TR_1 和 TR_2。

表 6-20 时态关系 TR_1

Name	Salary	Valid time
Peter	3500[2008,2011)	[2008,Now)
	4000[2012,2013)	
	6000[2014,2015)	
	7500[2016,Now	
Bush	3300[2009,2010)	[2009,Now)
	4500[2011,2013)	
	6000[2014,2015)	
	7600[2016,Now)	

表 6-21 时态关系 TR_2

Name	Title	Valid time
Peter	Lecture[2010,2011)	[2010,Now)
	Associate-Prof. [2012,2015)	
	Prof. [2016,Now)	
Bush	Lecture[2010,2011)	[2010,Now)
	Associate-Prof. [2012,2014)	
	Prof. [2014,Now)	

对于 Name 属性进行自然连接，以可连接元组有效时间的交作为连接后元组的有效时间，得到如表 6-22 所示连接结果。

表 6-22 自然连接结果

Name	Title	Salary	Valid time
Peter	Lecture[2010,2011)	3500[2008,2011)	[2010,Now)
	Associate-Prof. [2012,2015)	4000[2012,2013)	
	Prof. [2016,Now)	6000[2014,2015)	
		7500[2016,Now)	
Bush	Lecture[2010,2011)	3500[2009,2010)	[2010,Now)
	Associate-Prof. [2012,2014)	4500[2011,2013)	
	Prof. [2015,Now)	6000[2014,2015)	
		7600[2016,Now)	

Peter 元组 Salary 属性值的第一个数据有效时间[2008,2011)调整为其与元组有效时间[2010,Now)的交[2010,2011)。

Bush 元组 Salary 属性值的第一个数据有效时间[2009,2010)与元组有效时间[2010,

Now)交为空，该数据需要删除。

调整后即得到时态连接 $TR_1 \times TR_2$，如表 6-23 所示。

表 6-23　时态连接 $TR_1 \times TR_2$

Name	Title	Salary	Valid time
Peter	Lecture[2010,2011)	3500[2010,2011)	[2010,Now)
	Associate-Prof.[2012,2015)	4000[2012,2013)	
	Prof.[2016,Now)	6000[2014,2015)	
		7500[2016,Now)	
Bush	Lecture[2010,2011)	4500[2011,2013)	[2010,Now)
	Associate-Prof.[2012,2014)	6000[2014,2015)	
	Prof.[2015,Now)	7600[2016,Now)	

6.3　双时态关系数据模型

对于数据库而言，数据模型和相应查询语言彼此对应。现已形成标准的时态关系查询语言为 TSQL2，它由成立于 1993 年的 TSQL2 语言设计委员会在 1994 年正式颁布。作为时态数据模型及数据查询研究成果和 SQL92 的结合，TSQL2 实际上建立了一种双时态数据模型。按照问题的侧重点不同，TSQL2 双时态数据模型可分为以下两种主要形式。

(1) 双时态概念数据模型(Bitemporal Conceptual Data Model，BCDM)：面向查询和逻辑设计。

(2) 表示数据模型(Representational Data Model，RDM)：面向关系平台数据存储。

自 1994 年标准提出后，TDB 理论研究和实际中使用的数据模型通常都是 TSQL2 提出的 BCDM 及 RDM。

6.3.1　双时态概念数据模型

作为一种支持有效时间和事务时间的双时态概念模型，BCDM 出发点是实现双时态数据语义而非数据实际存储和用户表示，它在概念上与 RDB 保持兼容，使得 TDB 中的任意时间快照关系都是 RDB 中合法的数据文件。

BCDM 是指对关系元组添加双时态标签 T 而得到的数据结构。其中：

$$T=\{(t,v) \mid t\in[\mathrm{Ts},\mathrm{Te}),v\in[\mathrm{Vs},\mathrm{Ve})\}$$

(1) Ts 和 Te：Ts 为插入该元组时刻，即事务时间起始点；Te 为修改或删除该元组时刻。

(2) Vs 和 Ve：Vs 为元组生命周期起始时刻；Ve 为相应终止时刻。

(3) $t\in[\mathrm{Ts},\mathrm{Te})$ 和 $v\in[\mathrm{Vs},\mathrm{Ve})$：分别表示事务时间点 t 和有效时间点 v。

由此可知，BCDM 中元组的时间标签为分别表示事务时间和有效时间的双时态标签(timestamp)，而时间标签一般为 T-V 双时态平面上时间格点集合；而 BCDM 中的时态关系元组由一个常规关系元组和相应的双时态标签组成。

【例 6-6】　一个基于 BCDM 时态关系元组实例，如表 6-24 所示。

表 6-24　一个基于 BCDM 时态关系元组实例

Name	Title	Salary	Timestamp
Raul	Prof.	9200	{(2013-06-01,2013-06-01)…(2013-06-01,2013-08-31) (2013-06-02,2013-06-01)…(2013-01-01,2013-08-31) (2013-06-03,2013-06-01)…(2013-01-02,2013-08-31) … (uc,2013-06-01)… (uc,2016-08-31)}

在表 6-24 中,"Raul、prof.、9200"是常规关系元组,其后部分就是相应双时态标签。

如果在当前时刻查询该元组,则得到时间标签为图 6-7 所示 TT-VT 平面中阴影方框中所有时间格点组成的集合,而每个格点都是由事务时间点和有效时间点组成的二元组。

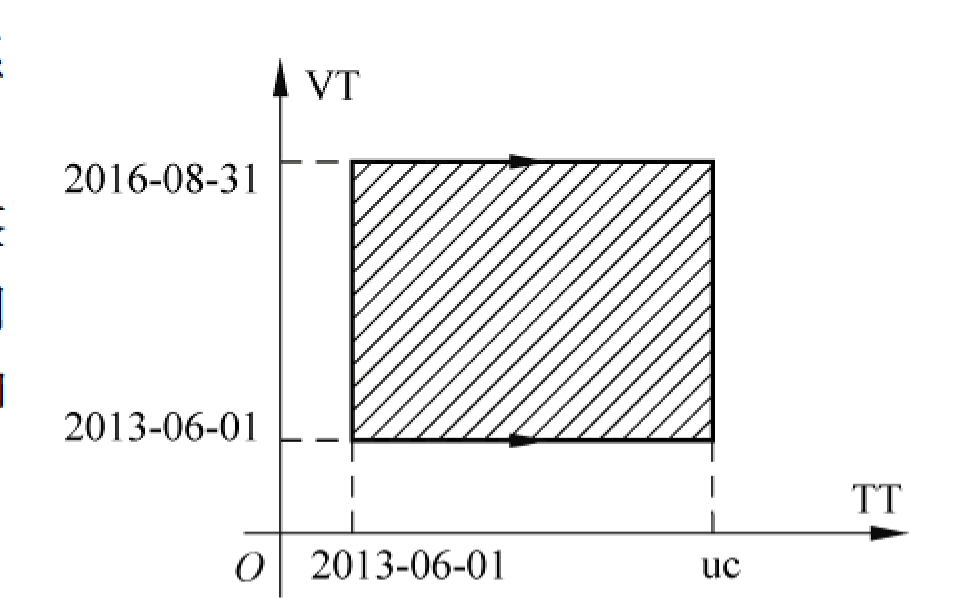


图 6-7　BCDM 双时态标签

BCDM 具有下述基本性质。

(1) 面向查询和逻辑设计。双时态标签结构简单,逻辑上可看作是以事务时间为索引的时间元素集合,便于查询和优化。

(2) 非时间属性取值原子性。BCDM 在概念和语义上与关系数据模型尽量保持兼容,除去时间标签,相应时态数据退化为关系数据,由于关系模型需要具有原子性,BCDM 不允许出现除时间标签外其他值都相等的元组。

(3) 递增到当前时刻的时间序列。事务时间中使用变量 uc,这是一个以给定时间粒度为单位的递增的时间变量,它生成一个从起始时间开始、逐渐递增且直到当前的时间序列。

(4) 时间数据取值非原子性。由于时间标签是时间元素(时刻二元组)集合,本身具有结构而非 1NF,因此基于 BCDM 时态关系元组中的时间标签不能称为时间"属性"。

6.3.2　表示数据模型

由于常规数据库一般并不支持区间取值运算,如要在[5,14)中取值 12,则需要将[5,14)中所有时间都进行存储,仅仅简单存储"5"和"14"是不够的,因此 BCDM 数据模式中双时态标签可以有效地用于查询处理;同时,由于可以简洁明确将时态元组表示为常规元组和双时态标签整合的形式,因此也方便于进行模式的逻辑设计。但是,由于 BCDM 模式中的 uc 线性时间递增,整个双时态标签表示会比较庞杂,对于用户使用而言不够直观和便捷。有鉴于此,Snodgrass 提出与 BCDM 能力等价的表示数据模型(Represent Data Model,RDM)。

RDM 数据模型:数据结构为形式 R(A_1,…,A_n;TTs,TTe,VTs,VTe)

(1) A_1,…,A_n,为常规关系数据属性。

(2) TTs、TTe 分别表示事务时间的起始时刻和终止时刻。

(3) VTs、VTe 分别表示有效时间的起始时刻和终止时刻。

此时,TTs、TTe 和 VTs、VTe 需要成对出现,表示相应事务时间和有效时间分别为时间期间[TTs,TTe)和[VTs,VTe)。通过给定时间粒度,就和确定相应时间期间内的任何

时间点。

表 6-23 中基于 BCDM 时态关系元组使用 RDM 表示后如表 6-25 所示。

表 6-25　基于 RDM 时态关系元组

Name	Title	Salary	TTs	TTe	VTs	VTe
Raul	Prof.	9200	2013-06-01	uc	2013-06-01	2016-08-31

【例 6-7】 表 6-26 是一个基于 RDM 时态关系实例。

表 6-26　一个基于 RDM 时态关系实例

Name	Title	Salary	TTs	TTe	VTs	VTe
Black	lecture	4000	2010-12-31	2012-12-31	2010-10-01	2012-12-31
Black	lecture	4300	2013-01-01	uc	2013-01-01	2014-05-31
Black	Associate-Prof.	6500	2014-06-30	2015-12-31	2014-06-01	2015-12-31
Black	Associate-Prof.	7500	2016-01-01	uc	2016-01-01	Now

与 BCDM 不同，RDM 中如果关系数据属性即(A_1，A_2，…，A_n)相同，而时间标签不同也看作不同元组。

设 Black 任 lecture 时，工资从 4000 元调整到 4300 元时间应从 2013-03-01 开始，而不是从 2013-01-01 开始，这个错误在 2013-03-01 修改数据时得到纠正。此时，表 6-26 中第 1 行改为如表 6-27 所示的第 1、2 行；表 6-26 中第 2 行改为如表 6-27 所示的第 3、4 行。其中，表 6-27 示的带有阴影的第 2 行和第 4 行为当前快照行。

表 6-27　修改后数据在 RDM 中的表示

Name	Title	Salary	TTs	TTe	VTs	VTe
Black	lecture	4000	2010-12-31	2012-12-31	2010-10-01	2012-12-31
Black	lecture	4000	2013-03-01	uc	2013-03-01	2014-05-31
Black	lecture	4300	2013-01-01	2013-03-01	2013-01-01	2014-05-31
Black	lecture	4300	2013-03-01	uc	2013-03-01	2014-05-31
Black	Associate-Prof.	6500	2014-06-30	2015-12-31	2014-06-01	2015-12-31
Black	Associate-Prof.	7500	2016-01-01	uc	2016-01-01	Now

RDM 具有下述性质。

(1) 时间标签原子性。TTs、TTe、VTs、VTe 是原子属性，满足 1NF 条件，像关系数据一样处理。

(2) 时间标签区分性。RDM 允许存在只有时间标签不同而常规关系元组相同的元组。

(3) 面向存储性。基于 RDM 时态关系数据可节省大量存储空间，但在查询处理时需进行适当处理，通常是转换为 BCDM 形式。

6.4　时间变量

当时态关系中时间期间的时间点尚未到达终点而实际应用场景中又需要不断单调增加时，就需要引入时间变量概念。时间变量分为有效时间变量 Now、事务时间变量 uc 和当前

时间变量 CT 3 种情形。时间变量在施行数据操作时需要和具体时间绑定。

(1) 当前时间变量 CT：进行实际数据操作的时间，与系统当前时间绑定。

(2) 事务时间变量 uc：进行数据事务操作的时间。

(3) 有效时间变量 Now：表示数据生命周期一直延续到“当前”或“现在”都还未结束。

CT 是系统当前时间的一种逻辑表示，uc 作为系统时间的事务时间终点可以由系统自动绑定；但 Now 由于事务时间相对于有效时间的技术“延迟性”而具有多种语义，实际数据操作绑定时需要进行比较细致的分析与处理。

本节与 6.5 节及 6.6 节所涉及的双时态关系都是基于 BCDM 的。

6.4.1 双时态关系的分析与解构

由于变量复杂语义涉及多种因素，因此有必要引入相关概念作为讨论的基础。

1. 时间无关与非时态属性

如前所述，在一个时态关系 TR 中，并不是所有属性都与时间(直接)相关。

设 A 是 TR 中一个属性。A 中所有属性值对应的有效时间期间集合称为 A 的有效时间集 Ω_A。$C_0=(\min\Omega_A, \max\Omega_A)$，其中 $\min\Omega_A$ 和 $\max\Omega_A$ 分别是 A 中的最小和最大时间端点。

f 是 A 的属性值域 V_A 到 Ω_A 上的映射 $f: V_A \rightarrow \Omega_A$。

(1) 时间无关属性：若对于 $\forall x \in V_A$，都有 $f(x)=C_0$，则称 A 为时间无关属性，否则称为时间相关属性。

例如，在时态关系(Name, Id, Title, Dept, Salary, VT, TT)中，Name 和 id 可以看作时间无关属性，Title、Dept 和 Salary 可以看作是时间相关属性。时态关系 TR 中可以存在多个时间无关属性，时间无关属性在一定环境中相对稳定，不随时间消长而变化。

(2) 时态与非时态属性：时态关系中的时间标签称为时态属性，非时间标签的属性为非时态属性。

时间无关属性是具有更强限制的非时态属性。

2. 主体实例与快照等价组

双时态关系相比于常规关系，其数据操作的头绪更为繁多。为了分析叙述上的明确和技术实现上的可行，需要更为精细地对双时态进行适当解构。为此可先将一般双时态关系分解为“主体实例”集合，再将主体实例分解为“快照等价组”的序列；同时在快照等价组中明确“当前版本”和“最新状态”元组的概念。

1) 主体实例

在时态关系 TR 选定一个时间无关属性 A，由 A 在 TR 上元组之间引入一个等价关系 ，即两个时态元组 T_1 和 T_2 具有关系 $T_1 \quad T_2 \Leftrightarrow \Pi_A(T_1)=\Pi_A(T_2)$，其中，$\Pi_A(T)$ 表示元组 T 在属性 A 上的投影。由此得到 TR 关于 的等价类集合$\{MI_1, MI_2, \cdots, MI_k\}$。

上述的 A 称为 TR 的主体属性；由 划分的等价类 $MI_j(1 \leqslant j \leqslant k)$ 称为 TR 关于 A 的一个主体实例。

【例 6-8】 设有时态关系 Personnel 如表 6-28 所示。

表 6-28 Personnel 双时态关系

Name	Dept	Position	Salary	VTs：VTe	TTs：TTe
Bob	Math	Vice-leader	6200	2014-01-01：Now	2014-01-10：2015-01-09
Bob	Math	Vice-leader	6300	2014-01-01：2004-12-31	2015-01-10：uc
White	IS	Leader	8500	2015-01-01：Now	2015-01-10：uc
Raul	CS	Leader	8500	2013-10-01：Now	2013-01-10：uc

在表 6-28 中，Name 是时间无关属性，前两个元组描述同一“主体”状态，构成一个主体实例；而第 3、4 两个元组分别是两个不同的主体实例。

本节以下讨论时态关系都是指主体实例。

2）快照等价组

给定一个主体实例，可将其中各个元组按照其非时态部分相同与否进行分组，由此得到的各个分组中的元组只有在时间标签上不同，这样的分组称为主体实例的快照数据等价组。

快照等价组（Snapshot Equivalent Group，SEG）：一个主体实例满足如下条件元组子集 $\{T_1, T_2, \cdots, T_k\}$，称为一个快照等价组，并记为 SEG。

（1）非时态相关属性值相同。

（2）事务时间满足后接关系：$TT(T_1) \leqslant TT(T_2) \leqslant \cdots \leqslant TT(T_k)$，其中 $TT(T_{i-1}) \leqslant TT(T_i)$ 等价于 $TTe(T_{i-1}) = TTs(T_i)(i=2,3,\cdots,k)$。

（3）至多有一个事务时间终点为 uc 的元组，若有则为 T_k。

主体实例中每一个 SEG 对应一条记录在数据库中的演变过程。

条件（1）限定组内记录的非时态相关属性值相同，按照快照数据库考虑就等同于一条记录。条件（2）限定组内记录的事务时间满足后接关系，为组内记录排序提供依据，而 T_k 可以看作是 SEG 的“最大元组”

3. 当前版本与最新状态

在双时态关系中，可以对任意元组进行查询，但只能针对特定元组进行更新。为了描述这种情形需要引入当前版本元组和最新状态元组概念。

1）当前版本元组

当前版本元组（current version tuple）：快照等价组 SEG 中的最后一个时态元组 T_k 记为 $T_{lat}(SEG) = T_k$，这是 SEG 中数据记录“最后”一次的演变结果。

当 $TTe(T_{lat}(SEG)) = uc$ 时，就称 $T_{lat}(SEG)$ 为时态关系当前版本元组（简称为当前版本）；否则就称为非当前版本元组（简称为非当前版本）。

对于非当前版本，其事务时间终点是一个固定时间点，这就表明该元组在数据库中已经被逻辑删除；对于当前版本，表明了该元组自插入 TR 后就没有进行逻辑上的数据删除和修改操作。由此可知，每个 SEG 至多只有一个当前版本；当一个主体实例具有多个 SEG 时，就可以具有多个当前版本。

2）最新状态元组

最新状态元组（latest state tuple）：若主体实例中的当前版本元组 T 满足 $VT_e(T) = Now$，则称 T 为主体实例的最新状态并记为 $T_{cur}(T_{lat}(SEG))$。也就是说，主体实例中满足

$VT_e(T)=Now \wedge TT_e(T)=uc$ 的元组就是其最新状态元组。

按照定义，最新状态一定是当前版本，并且由于 Now 语义限定，一个主体实例至多只能具有一个最新状态。

【例 6-9】 一个双时态关系如表 6-29 所示。

表 6-29 双时态关系

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Bob	Math	Vice-leader	6200	2014-01-01：Now	2014-01-10：2015-01-10
Bob	Math	Vice-leader	6200	2014-01-01：2014-12-31	2015-01-10：uc
Bob	Math	leader	8500	2015-01-01：Now	2015-01-10：uc
White	IS	Leader	8500	2013-01-01：Now	2013-01-10：2016-01-10
White	IS	Leader	8500	2013-01-01：2014-01-01	2016-01-10：uc
White	IS	Leader	8500	2014-02-01：2014-10-01	2016-06-10：uc
White	CS	Leader	8900	2016-01-01：Now	2016-01-10：uc
Raul	CS	Leader	8500	2013-10-01：Now	2013-10－10：uc

在表 6-29 中，前 3 个元组构成 Bob 主体实例；Bob 主体实例中的前两个元组构成一个 SEG，第 3 个元组自成一个 SEG。其中，第 2 个元组和第 3 个元组都是相应 SEG 的当前版本；同时，第 3 个元组是 Bob 主体实例的最新状态元组。

在表 6-29 中，第 4～7 个元组构成 White 主体实例，其中第 4～6 个元组虽然非时态属性值相等，但事务时间不满足后接关系，所以需要分为由第 4 及第 5 元组组成和由第 6 元组组成的两个不同的 SEG。其中，第 5 个和第 6 个元组是相应 SEG 的当前版本元组，第 7 个是 White 主体实例的最新状态元组。

在表 6-29 中，第 8 个元组自成一个 Raul 主体实例，这也是一个 SEG，该元组既是当前版本也是最新状态。

4. 双时态关系解构及相关语义

在引入上述概念基础上可以将一般双时态关系进行逻辑上的解构。

(1) 双时态关系分解为主体实例的集合，这相当于将关系表中的元组进行分组。每个主体实例就是具有相同时间无关属性(键值)时态元组集合。在逻辑上，双时态关系的相关讨论可以转化为在其每个主体实例上的讨论。

(2) SEG 是对主体实例的进一步解构，它由给定主体实例中具有相同非时间属性的元组组成。SEG 的意义在于在双时态关系数据分析过程中突出对时间要素的处理。主体实例是其所有 SEG 的序列，各个 SEG 按照其中“最大元组”中事务时间始点 TTs 大小排序。

(3) SEG 中的元组按照前述事务时间的后接关系排序而形成一个序列。其“最大”元组是该 SEG 中数据的“最后”演进状态。

① 如果“最大”元组是当前版本，则表明是对相关数据进行了修改更新；否则就是进行“删除”更新。

② 如果“最大”元组还是最新状态，则表明当前版本中的数据信息的一直到当前时刻都是有效的。

(4) 由于只能对于当前版本进行数据更新，因此，引入当前版本概念是在双时态关系中

明确突显时态更新操作的对象。由于常规关系查询的内容体现在非时间属性之中，因此引入关系实例和 SEG 实际上是在双时态数据管理原理分析上"屏蔽"了属于常规关系处理部分，而集中于相应的时态部分，并由此探讨相应的技术实现路线。

双时态关系表的解构如图 6-8 所示。

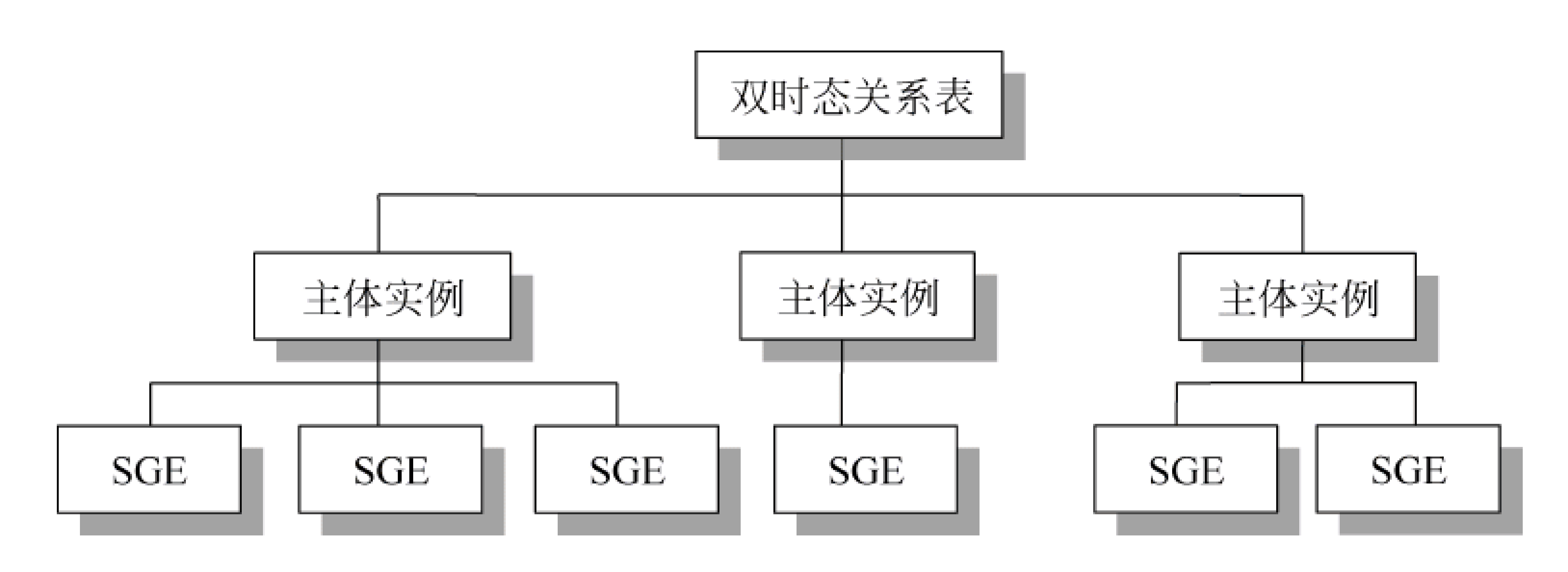


图 6-8　双时态关系表的解构

6.4.2　最新状态元组中 Now 语义处理

引入有效时间变量 Now 基本考虑是用于处理"时态元组的生命周期一直延续到现在都还未结束"这一语义问题。由于在实际数据处理过程中会出现"实际发生时间（有效时间）"与"数据入库时间（事务时间）"难以同步，即通常是后者"滞后"于前者的技术处理问题，因此会引发 Now 产生新的语义，使得在数据操作绑定 Now 时需要进行比较细致的分析与处理。

Now 只能出现在 SEG 的最新元组和非当前版本元组当中，需要分别讨论。先分析最新状态元组中的情形。

1. Now 语义分析

时态数据库中有效时间期间[VTs，VTe)可根据时间终点 VTe 分为两类。

(1) VTe＝确定时刻。此时表示相应数据的生命周期或者在"过去"已经结束（VTe＜CT），或者表示将在"未来"某个时刻结束（VTe＞CT）。

(2) VTe＝不确定时刻。此时表示数据有效时间随时间流逝而单调增加，其不确定性含义为："数据的有效时间截止当前时间还未结束，但不能确定在将来何时结束。"

研究不确定性端点时刻的有效时间处理机制是整合 TDB 中时间标签处理的一个基础课题，解决此类随时间变化有效时间端点时刻通常有两种处理方式。

(1) 常量方式。将不确定端点作为常量进行处理。此时有两种实现路线。

① 最大值方法：将 VTe 看作系统允许最大时间，如 IBM 的 DB2 中最大时间为 8000 年。

② 变动常量方法：将 VTe 看作变量，每天（假设时间粒度是"天"）将 VT_e 取作当天时间 CT。

最大值方法使用简单，但不能体现"不确定"含义，与人们日常直观相悖，因为系统最大时间很大，数据实际有效期不可能具有如此长的时间。此外，如果以系统最大时间为 VTe 变动，则相应数据需更新时，数据库会就有可能出现两个相互矛盾的数据文件，破坏数据库一致性。

变动常量方法能够体现"不确定"概念，但不符合常规技术处理要求，因为在实际情形中不需要也不可能每天更新。此外，在应用中由于确定和不确定的 VTe 都是"常定"形态的时

刻，系统运行中需要分清哪些是随着时间需要逐日更新，哪些不需要逐日更新，增加了技术实现上复杂性。

(2) 变量方式。将不确定端点记为某个变量符号如 Now，对数据进行操作时再对变量具体取值。此时也有两种实现路线。

① 当前时间语义变量方法：选用 Now 作为有效时间变量主要是考虑到 Now 具有 at the present time 的自然语义。在数据操作过程中，设计一种机制将 Now 与操作当前时间 CT 绑定以实现变量 Now 取值。这种变量时间设计称为 Now 相关时间，它既能够体现"不确定"含义，因为"变动"的端点使用变量更为自然直观，还可以避免常量方法带来的缺陷和不足。

② 一般时间语义变量方法：即结合时态数据库实际运行情况，赋予变量较为广泛的语义。

2. TDB 框架中 Now 语义

在变量 Now 的时态数据库运行当中，实际问题并不是像原先引入 Now 时那样简单。

(1) 有效时间概念语义问题。当 VTe＝Now 时，如果采用当前时间语义将 Now 和 CT 绑定，按照有效时间概念，严格来说，其表示相应数据只在从过去某个时刻到"今天"有效，明天就不会有效，这有悖于常识，在实际语义操作处理中带来不便，这可以看作是一种悲观主义态度。

(2) 事务时间关联语义问题。在使用 Now"当前"语义过程中，实际上隐含一个相当强的假设，即数据在现实世界中开始有效或改变的时刻和该数据创建或更新的事务时刻"同步"，这与有效时间和事务时间的"正交性"矛盾，在数据库实际运行中难以做到。对于数据创建和更新来说，数据入库时刻相对数据实际生效或改变时刻通常都有一个"错位"情形：$VTs \neq TTs$，即错位量 $\Delta = TTs - VTs \neq 0$。

实际上，按照事务时间和有效时间的概念内涵不难推知。

① 当 $VTs < TTs$ 即 $\Delta > 0$，Now 实际上可能表示过去时间。

② 当 $TTs < VTs$ 即 $\Delta < 0$，Now 实际上可以表示将来时间。

变量 Now 引进，不仅可以解决 VTe 为不确定时间问题，而且还揭示出双时态数据库运行过程中的某些具有研究意义和技术价值的课题。

【例 6-10】 设有表 6-30～表 6-32 表示的 3 个时态关系，QT 表示所需要查询数据的时间约束，通常假设为时间期间或时间点。

在表 6-30 中，假设系统中数据更新在"事实"生效后第三日执行，即 $\Delta = 3$。如果 John 在 2016-08-31 离职，数据库中相应数据的有效时间终点 VTe 在事务时间 2016-09-03 时才能"物理"地改变为 2016-08-31。若有关部门在 CT＝2016-09-02 需要查询统计截至 QT＝2016-09-01 全体在职员工的情况，这里，QT＝2016-09-01∈[2016-08-29：2016-09-02)＝[CT－Δ：CT)。此时，如果将 Now 简单绑定于 CT＝2016-09-02，则对于 White 就会得到与事实不符的结果。如果将 Now 绑定于某个过去时间如 CT－Δ＝2016-08-29 就会得到合适结果。可见，此时 Now 不表示 current time 而具有过去时间 CT－Δ 语义。

表 6-30 John 时态关系

Name	Title	VTs：VTe
John	Peofessor	2016-03-01，Now

在表 6-31 中，假设有关部门做当年(2016)单位职工奖励工资预算，需要在当前时间 CT＝2016-10-01 查询 Black 本年度(QT＝2016-12-31)是否在职。假设 Black 的有效预测期(合同期)为 3 年。这里，QT＝2016-12-31∈[2016-10-01：2018-07-31)＝[CT：2018-07-31)。此时，将 Now 绑定于 CT 也是不合适的，可以考虑将 Now 绑定于“未来”某个时间，如将 Now 绑定于 2018-07-31 也许比较合适。

表 6-31　Black 时态关系

Name	Title	VTs：VTe
Black	Manager	2015-08-01，Now

设 CT＝2016-06-01，表 6-32 表示毕业生 White 已经和单位签订了合同(合同期三年)并且相应记录也进入了数据库，其中 VTs(L)和 VTe(L)分别表示合同 L 的逻辑有效时间始点和终点。但相关部门可能在 CT 查询截止 2016 年年底在职员工情形时。此时，从直观上考虑，Now 可以考虑绑定于“未来”时间：VTe(L)＝2019-06-30。

表 6-32　White 时态关系

Name	Title	VTs(L)：VTe(L)
White	Lecture	2016-07-01，Now

3. 确定和不确定查询

上述 Now 语义实际上包含了数据查询过程中可能具有的不确定性。实际上，前述 VTe＝Now 时间语义实际上可在两个层面进行分析。

(1) 确定性层面：即“从过去某个时刻到当前时刻是有效的”。

(2) 非确定性层面：即“有效期何时终结难以确定”。

在数据操作过程中 Now 需要和具体时间值绑定，而不同层面含义导致引起变量 Now 值的不同绑定，从而导致查询结果的事实确定性和事实不确定性。

1) VTs＜TTs

此时，表明错位量 Δ＝TTs－VTs＞0，表现事务时间相对于有效时间的滞后期。按照前述语义分析，相对于当前时刻 CT，Now 可能需要绑定于某个“过去”时刻。例如，可考虑将 Now 绑定于“CT－Δ”。此时，根据查询时刻 QT 分为下述情形如图 6-9 所示。

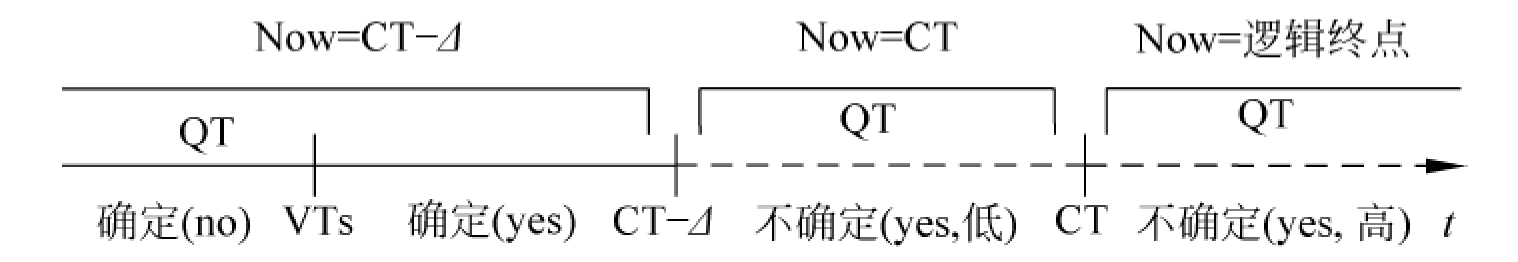


图 6-9　Now 绑定和查询结果输出

(1) 确定性结果。当 QT≤CT－Δ 时，而 Now 绑定于 CT-Δ，其中，

QT＜VTs：输出确定查询结果(no，无数据输出)；

VTs≤QT＜CT－Δ：输出确定查询结果(yes，有数据输出)。

(2) 非确定性结果。

① CT－Δ＜QT≤CT：Now 绑定于 CT，输出不确定查询结果(yes，有具较低不确定性

的数据输出)。

② CT<QT:Now 绑定于数据的逻辑有效预测期终点,输出不确定查询结果(yes,有具渐高不确定性的数据输出)。

上述 4 种情况如图 6-9 所示。

2) TTs<VTs

此时,表明了相关数据有效期还未实际开始但已经进入了数据库,对于这类只是逻辑上有效的数据,其查询结果不可避免带有不同程度上的非确定性,主要用于对状态或情形的预测。对于数据 d 来说,此时可以设 $\Omega=VTe(d)-VTs(d)$ 为数据的逻辑有效期,而$[VTs(d),VTe(d))=[VTs(d),VTs(d)+\Omega)$表示数据逻辑上的预测有效时间期间,例如一个即将生效的员工有效合同期。对于当前时间 $CT<VTs(d)$而言,此时 Now 具有"将来"时间语义,此时,可以考虑将 Now 与 $VTe(d)=VTs(d)+\Omega$ 绑定。根据查询时间 QT 不同情形,可以有具有确定性或非确定性的输出结果如图 6-10 所示。

(1) $QT<VTs(d)$:输出确定查询结果(no,无数据输出)。

(2) $VTs(d)\leqslant QT\leqslant VTs(d)+\Omega$:输出非确定性预测查询结果(yes,有数据输出,非确定性渐高)。

(3) $VTs+\Omega<QT(d)$:输出确定查询结果查询(no,无数据输出)。

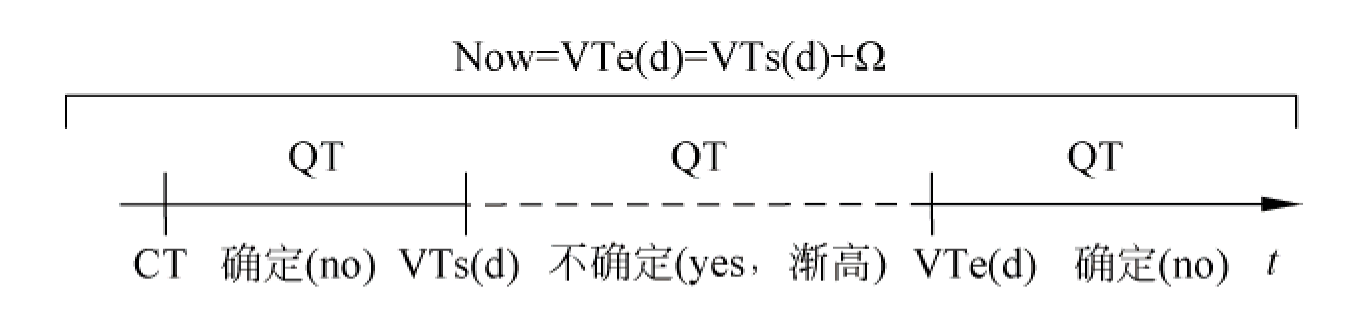


图 6-10 Now 将来语义绑定和结果输出

6.4.3 非当前版本 Now 语义处理

对于给定 SEG 中,非当前版本中 Now 表示非当前时刻语义现象更为常见。设元组 T_i 的时态标签中出现时态变量 Now,T_i所在的 SEG 为 S,同一 SEG 的其他元组为 $T_1,\cdots,T_{i-1},T_i,\cdots,T_k$,满足 $TT(T_1)\leqslant\cdots\leqslant TT(T_i)\leqslant\cdots\leqslant TT(T_k)$。对于任意 i 满足 $1\leqslant i\leqslant k$,此时 T_i 存在如下两种情形。

(1) 当 $i<k$ 时,T_i在 SEG 中其后有"后继"记录。这主要发生在插入相应新记录后的情形。

(2) 当 $i=k$ 时,T_i是所在 SEG 的最大元组。这主要发生在记录的"逻辑"删除情形。

1. 非最大元组中 Now 绑定

在给定 SEG 中非当前版本 T_i之后有"后继"记录即"最大者" T_k,这表明对原先的当前元组进行了更新,插入了新的元组。此时,Now 需和相应 SEG 中的"最大后继"元组 T_k中有效时间终点 VTe 绑定,即 $Now=VTe(T_k)$。

【例 6-11】 设有单个元组构成的"Raul"SEG 如下:

"Raul,CS,Assistant. prof,7500,[2015-01-01:Now;2015-01-10:uc)"

现在需要在 CT=2016-01-01 时向其中插入新的元组:

"Raul,CS,Assistant. prof,8900,[2016-01-01:Now;2016-01-10:uc)"

更新后的主体实例具有如表 6-33 所示的分别由第 1 和第 2 元组构的及由第 3 元组构

成的 SEG1 和 SEG2。

表 6-33　关于 Raul 的主体实例(一)

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Raul	CS	Assistant. Prof.	7500	2015-01-01：Now	2015-01-10：2016-01-09
Raul	CS	Assistant. Prof.	7500	2015-01-01：2015-12-31	2016-01-10：uc
Raul	CS	Prof.	8900	2016-01-01：Now	2016-01-10：uc

设 CT=2016-02-28,在 SEG1 中进行如下双时态查询以检索 Raul 的基本情况：

TT(Q)=(2015-09-01：2015-11-01)，VT(Q)=(2016-02-20：2016-02-20)

此时,表 6-33 中的第一个元组不是最新状态,Now 也不能绑定于当前时间“CT=2006-02-28”,否则输出如下结果：

“Raul,CS,Assistant. prof,7500,(2016-02-20：2016-02-20；2015-09-01：2015-11-01)”

而这与如下的实际结果矛盾：

“Raul,CS,prof,8900,(2016-02-20：2016-02-20；2016-01-10：2006-02-28)”。

事实上，Now 应与 SEG1 中的“最大”元组,即表 6-33 中第 2 个元组(也是当前版本元组)的有效时间终点 VTe=2015-12-31 绑定,即 Now VTe=2015-12-31,绑定后 SEGI 不是查询结果,与实际情况相符。

2. 最大元组中 Now 绑定

SEG 中元组 T_k没有“后继”记录,这主要发生在记录被“逻辑”删除的情形。此时,存在着两种“语义”情形。

1) T_k所在 SEG 存在着“后继” SEG1

此时表达的语义是对 T_k 进行了修改,SEG′是修该后数据组成的新的 SEG。

设后继 SEG1 中最大后继元组为 T′,此时 Now 绑定于 VTs(T′)−1 即 Now=VTs(T′)−1。

【例 6-12】 如表 6-34 所示,Raul 主体实例中第 1 个元组组成一个 SEG,表示一个已经被“逻辑”删除的记录。其后继 SEG1 由第 2 个元组组成,是对第 1 个元组进行修改后的元组。SEG1 的“最大”元组 T′就是图 6-40 中第 2 个元组。如果需要查询满足 TT(Q)=(2015-02-01：2015-07-31),VT(Q)=[2015-03-01：2015-09-30)的数据,此时 Now 就与 VTs(T′)−1=2014-12-31 绑定。输出结果将为空,符合实际情况。

表 6-34　关于 Raul 的主体实例(二)

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Raul	CS	Assistant. Prof.	7500	2015-01-01：Now	2015-01-10：2016-01-09
Raul	CS	Assistant. Prof.	7800	2015-01-01：Now	2016-01-10：uc

2) T_k所在 SEG 没有“后继”SEG′

此时,T_k为相应主体实例中最后一个元组,表达的语义是该元组已经被逻辑删除。直接将 Now 绑定于 CT 即 Now=CT。

【例 6-13】 如表 6-35 所示的元组已经被逻辑删除,其后没有后继 SEG1,Now 与当前时间 CT 绑定。

表 6-35　关于 Raul 的主体实例(三)

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Raul	CS	Assistant. Prof.	7500	2016-09-01：Now	2016-05-10：2016-06-10

6.5　双时态数据操作

双时态关系数据操作可以分为数据查询和数据更新两个部分进行讨论。

6.5.1　双时态数据查询

具有变量的数据库中数据查询前提是确定或绑定有效时间变量 Now 值。如前所述,这种绑定与 Now 语义相关,需要对 Now 的实际应用情况进行分析。

1. 时态连接运算

时态关系连接除了满足常规连接条件之外,还需要满足时态连接条件。

可时态连接元组：时态关系表 TR_1 中的元组 T_{10} 和时态关系 TR_2 中的元组 T_{20} 称为是可进行时态连接的(简称可时态连接的),如果满足以下条件。

(1) T_{10} 和 T_{20} 处于同一版本当中。

(2) T_{10} 和 T_{20} 除去时间标签之后,作为常规关系元组是可以连接的。

(3) T_{10} 和 T_{20} 相应的有效时间期间交叠(overlap)。

由于连接过程并不显式表示关于时间的选择,此时,Now 绑定可以不考虑查询时间 QT 的因素。在验证(3)时有下述两种情形。

① 当两个需要连接元组 T_{10} 和 T_{20} 中都出现 Now 时,将 Now 绑定于 CT,连接后有效时间终点恢复为 Now。

② 当需要连接元组中只有一个出现 Now 时,即 $VTe(T_{10})=Now \wedge VTe(T_{20})\neq Now$。若 $VTe(T_{20})\leqslant CT$,则 $VTe(T_{10})=Now$ 绑定于 CT；若 $CT<VTe(T_{20})$,则 $VTe(T_{10})=Now$ 绑定于 $VTe(T_{20})$。连接后时态元组有效时间终点 $VTe(VT(T_{10})\cap VT(T_{20}))$ 恢复为 Now。

时态关系连接过程可以分析如下。其中,v(T)表示时态元组的非时态部分即常规元组。

设主体实例 MI_1 和 MI_2 中两个时态元组分别为 $T_{10}=(v(T_{10}),VT(T_{10}),TT(T_{10}))$ 和 $T_{20}=(v(T_{20}),VT(T_{20}),TT(T_{20}))$，$MI_3$ 为连接后的新的主体实例。当 T_{10} 和 T_{20} 满足以下条件。

(1) $TT(T_{10})\subseteq TT(T_{20}) \vee TT(T_{20})\subseteq TT(T_{10})$。

(2) $VT(T_{10})\cap VT(T_{20})\neq\varnothing$。

(3) 非时态部分作为常规关系元组是可联接的。

定义时态连接算子 $\bowtie$ 定义为：$\bowtie(T_{10},T_{20})=\{(v(T_{30}),VT(T_{30}),TT(T_{30}))\}$,其中 $v(T_{30})=v(T_{10})\bowtie v(T_{20})$,$VT(T_{30})=VT(T_{10})\cap VT(T_{20})$,$TT(T_{30})=\min(TT(T_{10}),TT(T_{20}))$。

【例 6-14】 设有如表 6-36 和表 6-37 所示的两个时态关系 TR_1 和 TR_2,当前时间 CT=2016-05-01,讨论其时态连接。

表 6-36　时态关系 TR_1

Name	Title	VTs:VTe	TTs:TTe
Green	Manager	2013-01-01:2013-12-31	2015-01-10：uc
Green	Sr. Manager	2014-01-01:Now	2015-01-10：uc

表 6-37　时态关系 TR_2

Name	Salary	Dept	VVTs：VTe	TTs：TTe
Green	6800	CS	2014-01-01：2014-12-31	2015-01-10：uc
Green	8000	CS	2015-01-01：Now	2015-01-10：uc

这里，$TR_1=\{T_{11},T_{12}\}$和 $TR_2=\{T_{21},T_{22}\}$中元组都是同一版本；非时态部分是可连接元组；但只有 T_{12}中的第二个元组有效时间与 T_{21}和 T_{22}有效时间期间有交叠。

由于 $VTe(T_{21})\leqslant CT$，因此 $VTe(T_{12})=Now$ 绑定于 CT，此时 $VT(T_{12})=[2014\text{-}01\text{-}01,2016\text{-}05\text{-}01)$、$VT(T_{21})=[2014\text{-}01\text{-}01,2014\text{-}12\text{-}31)$、$VT(T_{12})\cap VT(T_{21})=[2014\text{-}01\text{-}01:2014\text{-}12\text{-}31)$。

由于 $VTe(T_{12})=VTe(T_{22})=Now$，两者都绑定于 CT，此时 $VT(T_{12})\cap VT(T_{22})=[2005\text{-}01\text{-}01:CT)$，最终输出结果中再恢复为 $VT(T_{12})\cap VT(T_{21})=[2005\text{-}01\text{-}01:Now)$作为连接后元组有效时间期间。连接运算结果如表 6-38 所示。

表 6-38　连接运算结果

Name	Salary	Dept	Title	VTs：VTe	TTs：TTe
Green	6800	CS	Sr Manager	2014-01-01:2014-12-31	2015-01-10:uc
Green	8000	CS	Sr. Manager	2015-01-01:Now	2015-01-10:uc

2. 时态投影运算

在投影运算中，可能会出现非时态部分相同而时态标签不同的时态元组，此时需要根据情况对元组进行时态归并处理，即对相应的时态标签进行“归并”。归并基本思想是先对投影结果快照组中的元组进行事务时间归并，再对同一版本中的元组进行有效时间归并。

(1) 事务时间归并。同一 SEG 中两个元组 T_1 和 T_2成立 $TT(T_1)\cap TT(T_2)\neq\varnothing$时，则称 T_1和 T_2是关于事务时间可归并的，此时将 T_2和 T_1的事务时间调整为 $TT=[\max\{TTs(T_1),TTs(T_2)\}:\min\{TTe(T_1),TTe(T_2)\})$。

(2) 有效时间归并。对于 SEG 中调整事务时间后的同一版本中元组再考虑进行有效时间归并。设 T_1和 T_2是如此的元组，则实行下述步骤。

步骤 1：Now 绑定

① T_1和 T_2是当前版本，如果存在 Now，则 Now 绑定于 CT。

② T_1和 T_2是非当前版本，如果 $T_1(T_2)$存在 Now，而 TT 是相应事务时间，则 $VTe(T_1)(VTe(T_2))=Now$ 绑定于 $TTs-1$。

步骤 2：时间归并

当 $VT(T_1)\cap VT(T_2)\neq\varnothing$、$VTe(T_1)=VTs(T_2)$或 $VTe(T_2)=VTs(T_1)$时，归并后有效时间期间为$[\min\{VTs(T_1),VTs(T_2)\}:\max\{VTe(T_1),VTe(T_2)\})$。

【例 6-15】 设有如表 6-39 所示双时态关系。

表 6-39 双时态关系 TR

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2013-01-01：Now	2013-01-10：2014-01-10
Black	EN	Engineer	7500	2013-01-01：2013-12-31	2014-01-10：uc
Black	EN	Sr. Engineer	8500	2014-01-01：Now	2014-01-10：2015-01-10
Black	EN	Sr. Engineer	8500	2014-01-01：2014-12-31	2015-01-10：uc
Black	EN	Sr. Engineer	9000	2015-01-01：Now	2015-01-10：uc

如果对 Name 和 Dept 进行投影的时态关系如表 6-40 所示。

表 6-40 在 Name 和 Dept 上的投影

Name	Dept	VTs：VTe	TTs：TTe
Black	EN	2013-01-01：Now	2013-01-10：2014-01-10
Black	EN	2013-01-01：2013-12-31	2014-01-10：uc
Black	EN	2014-01-01：Now	2014-01-10：2015-01-10
Black	EN	2014-01-01：2014-12-31	2015-01-10：uc
Black	EN	2015-01-01：Now	2015-01-10：uc

第 1 个和第 3 个元组事务时间不能进行调整，属于不同版本；其余元组属于当前版本，事务时间 TT=[2015-01-10：uc)，归并后有效时间 VT=[2013-01-01：Now)。归并后时态关系如表 6-41 所示。

表 6-41 事务时间和有效时间归并

Name	Dept	VTs：VTe	TTs：TTe
Black	EN	2013-01-01：Now	2013-01-10：2014-01-10
Black	EN	2014-01-01：Now	2014-01-10：2015-01-10
Black	EN	2013-01-01：Now	2015-01-10：uc

3. 时态选择运算

当查询是关于时间元素的选择时，需要将 Now 值“绑定”。

【例 6-16】 设有如表 6-42 所示的时态关系 Employee，其中错位量 Δ=10。

表 6-42 双时态关系 TR

Name	Dept	Position	Salary	VTs：VTe	TTs：TTe
Bob	Math	Vice-leader	6200	2014-01-01：Now	2014-01-10：2015-01-09
Bob	Math	Vice-leader	6300	2014-01-01：2004-12-31	2015-01-10：uc
White	IS	Leader	8500	2015-01-01：Now	2015-01-10：uc
Raul	CS	Leader	8500	2013-10-01：Now	2013-01-10：uc

(1) 查询 QT：TT(QT)=2014-06-30 ∧ VT(QT)=2014-05-25 的事件(Event)信息，由于非当前版本中只有表 6-42 中所示的第一个元组 T1，因此 VTe(T1)Now 绑定于 TTs(T2)−1=2014-01-09，其中 T2 为 T1 的后继元组。此时 VT(QT)=2014-05-25∈VT(T1)=[2014-01-01：2015-01-09)，由此得到相应查询结果为 T1。

(2) 假设 CT=2015-04-20,查询 QT:TT(QT)=uc∧VT(QT)=[2015-02-15:2015-03-15)的信息,由于 VTe(QT))=2015-03-15 小于 CT−Δ=2015-04-10,表 6-42 中的第 3 个元组 T3 中 Now 与当前时间 CT 绑定,即 VTe(T3)=Now=CT=2015-04-20,由此得到所需要信息。

(3) 如果需查询有效时间期间 VT(QT)=[2015-02-15:2015-04-15)情形。由于 CT−Δ≤VTe(QT)≤CT,即 VTe(QT)=2015-04-15∈[CT−Δ:CT)=[2015-04-10:2015-04-20),VTe(T3)=Now 绑定于 CT−Δ=2015-04-10,此时查询输出不确定信息。

6.5.2 双时态数据更新

双时态关系数据更新只能针对当前版本,相应的插入、删除和修改操作只能在当前版本集合上进行。

1. 插入更新

插入更新分为下面两种情形。

(1) 插入后构成一个新主体实例。此时插入的元组就是一个最新状态元组。

(2) 插入到已有主体实例当中。此时插入过程与主体实例当前版本元组相关。

① 若当前版本不是最新状态元组,直接插入新的元组作为最新状态元组。

② 若当前版本是最新状态元组,需要将该最新状态元组进行两次处理:首先,将最新状态元组中 uc 修改为 CT 后作为历史记录予以保留;然后,再将最新状态元组中 Now 修改为插入元组中 VTs−1 后作为“新生”当前版本元组插入存储;最后,再将需要新插入的元组作为新的最新状态元组插入存储。

【例 6-17】 设有 Personel 工资关系 TR(Name,Dept,Title,Salary;VTs:VTe,TTs:TTe),时间错位量 Δ=9(天)。设有表 6-43 所示 Jake 主体实例,其中当前版本元组非最新状态。

表 6-43 Jake 主体实例中无最新状态元组

Name	Dept	Title	Salary	VTs:VTe	TTs:TTe
Jake	CS	Lecture	7500	2013-01-01:2015-12-31	2016-01-10:uc

插入新元组后如表 6-44 所示。

表 6-44 Jake 主体实例中插入新元组

Name	Dept	Title	Salary	VTs:VTe	TTs:TTe
Jake	CS	Lecture	7500	2013-01-01:2015-12-31	2016-01-10:uc
Jake	CS	Associate-Prof.	8500	2016-09-01:Now	2016-09-10:uc

【例 6-18】 Black 在 VTs=2017-01-01 进入 EN 部门工作,担任 Engineer,工资为 7500,那么他的记录在 TTs=2017-01-10 被录(插)入到相应双时态关系表中,由此建立起 Black 主体实例如表 6-45 所示。

表 6-45 插入新元组建立 Black 主体实例

Name	Dept	Title	Salary	VTs:VTe	TTs:TTe
Black	EN	Engineer	7500	2017-01-01:Now	2017-01-10:uc

当 Black 在 2017-08-01 升职并加工资，数据库记录在 2017-08-10 进行更新。

首先，将原来记录作为历史记录保留，此时系统自动绑定原记录中事务时间的终止时刻 uc 为 CT(系统时间)，即是 2017-08-10，表示该记录这一状态的事务在该时刻终止。

其次，将原记录中 Now 绑定为 VTs－1＝"2017-07-31"，TTs 修改为 CT＝2017-08-10，并将元组(Black，EN，Engineer，7500，2017-01-01，2017-07-31，2017-08-10，uc)作为一个当前版本记录保存，此时和历史记录一起构成 Black 实例的一个 SEG，如表 6-46 所示。

表 6-46　保存历史记录后得到的 SEG

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2017-01-01：Now	2017-01-10：2017-08-10
Black	EN	Engineer	7500	2017-01-01：2017-07-31	2017-08-10：uc

最后，插入新记录：(Black，En，Sr Engineer，8500，2017-08-01，Now，2017-08-10，uc)，该记录作为当前元组存储同时构成一个由最新状态组成的 SEG，如表 6-47 所示。

表 6-47　插入新元组后得到的主体实例

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2017-01-01：Now	2017-01-10：2017-08-10
Black	EN	Engineer	7500	2017-01-01：2017-07-31	2017-08-10：uc
Black	EN	Sr. Engineer	8500	2017-08-01：Now	2017-08-10：uc

在表 6-47 中，第 1、2 两条记录描述了 Black 同一状态的两个不同版本，第 1 条记录事务时间的终止时刻表明是该记录是历史版本，第 2 条记录是相应 SEG 的当前版本。第 3 个元组是新插入的记录，Now 和 uc 的同时存在说明该元组是"最新状态"。

再假设 Black 在 2018-01-01 晋升工资为 9000，数据库在 2018-01-10 进行更新。按照上述同样的步骤得到结果如表 6-48 所示。其中，第 1 个和第 2 个元组、第 3 个和第 4 个元组及第 5 个元组分别构成了 3 个 SEG。

表 6-48　再次插入新元组后得到的主体实例

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2017-01-01：Now	2017-01-10：2017-08-10
Black	EN	Engineer	7500	2017-01-01：2017-07-31	2017-08-10：uc
Black	EN	Sr. Engineer	8500	2017-08-01：Now	2017-08-10：2018-01-10
Black	EN	Sr. Engineer	8500	2017-08-01：2017-12-31	2018-01-10：uc
Black	EN	Sr. Engineer	9000	2018-01-01：Now	2018-01-10：uc

从以上 Black 历史记录演变过程可以看出，此时 Now 实际上是按照"过去"时间绑定，即 Now 具有过去时间语义。另外，由于双时态要求，需要同时"插入"两条记录，分别为原有数据的最新版本和新添加记录的当前版本，原有数据作为历史旧版本仍然保留在数据库中。由此可知，相对于有效时间数据库，基于变量的双时态数据库的数据操作技术更为复杂和精细。

2. 删除更新

双时态关系删除只能针对当前版本，因此只需要根据删除条件在当前版本集合中选择

出满足条件的元组，然后将其中的 uc 与 CT 绑定后作为历史记录予以保留即可。

【例 6-19】 删除表 6-48 所示主体实例中满足条件 Salary＝8500 的元组，其中 CT＝2018-03-01。删除操作后的主体实例如表 6-49 所示。

表 6-49　删除满足条件 Salary＝8500 的当前版本后得到主体实例

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2017-01-01：Now	2017-01-10：2017-08-10
Black	EN	Engineer	7500	2017-01-01：2017-07-31	2017-08-10：uc
Black	EN	Sr. Engineer	8500	2017-08-01：Now	2017-08-10：2018-01-10
Black	EN	Sr. Engineer	8500	2017-08-01：2017-12-31	2018-01-10：2018-03-01
Black	EN	Sr. Engineer	9000	2018-01-01：Now	2018-01-10：uc

3. 修改更新

修改操作实际上是删除相应的原当前版本，将已经更新的元组作为新的当前版本插入到时态关系当中，即分别施行前述的删除与插入算法。

【例 6-20】 如果 CT＝2018-03-10 需要将表 6-48 中元组 T_{k0} 修改为 T_{k1}，其中：

T_{k0}＝(Black，EN，Engineer，7500，2017-01-01：2017-07-31，2017-08-10：uc)

T_{k1}＝(Black，EN，Engineer，7800，2017-01-01：2017-07-31，2018-03-10：uc)

按照上述算法，得到修改后时态关系如表 6-50 所示。类似也可以讨论对于有效时间进行的修改。

表 6-50　修改元组后得到主体实例

Name	Dept	Title	Salary	VTs：VTe	TTs：TTe
Black	EN	Engineer	7500	2017-01-01：Now	2017-01-10：2017-08-10
Black	EN	Engineer	7500	2017-01-01：2017-07-31	2017-08-10：2018-03-10
Black	EN	Engineer	7800	2017-01-01：2017-07-31	2018-03-10：uc
Black	EN	Sr. Engineer	8500	2017-08-01：Now	2017-08-10：2018-01-10
Black	EN	Sr. Engineer	8500	2017-08-01：2017-12-31	2018-01-10：2018-03-01
Black	EN	Sr. Engineer	9000	2018-01-01：Now	2018-01-10：uc

由上述讨论过程实际可以得到如下的双时态关系运算的封闭性定理。

【定理 6-1】 基于变量的双时态关系进行上述时态查询与更新操作后得到结果依然是一个双时态关系。因此，数据查询与更新操作都是双时态关系集合到自身的映射。

6.6　时态关系数据语言 TSQL2

TSQL2 是基于 BCDM 的双时态数据库语言，基本要求是时态关系表中不能含有非时态部分相同而时间标签不同的元组，即一个快照只能具有一个时间标签，因此常规关系元组中的时间标签通常为时间元素。

下面以一个医疗数据库为实例简要介绍 TSQL2 的基于常规 SQL 的时态扩展基本特点。医疗实例数据库由“处方”和“诊断”两个时态关系表组成，其中“处方”关系表记录医生给病人开的医疗处方，语义约束是每种处方只开一种药；“诊断”关系表记录医生给病人所

做的检查,而检查作为一种“事件”总是发生在某一时刻。

6.6.1 双时态关系数据创建

作为一种关系型数据语言,TSQL2 需要兼容常规关系数据操作;而作为一种双时态数据语言,TSQL2 还需要能够处理单时态的各种情形。基于上述考虑,TSQL2 将关系型的基本表分为下述 4 种类型。

(1) 快照关系。其中关系元组无时间标签,即为常规关系表。

(2) 有效时间关系。其中关系元组具相应的有效时间标签。此时,按照时间标签对应的实际情形又分为“状态”和“事件”两种类型。

① 有效时间状态关系:用以表示实体所处的状态(State),时间标签为对象所处状态的有效时间且通常表示为时间元素,由子句 AS VALID[STATE]标识,其中 STATE 可选且为默认。

② 有效时间事件关系:用以表示实体发出的事件(Event),时间标签表示事件发生的确定时刻且通常为时间点的集合,由子句 AS VAIID EVENT 标识。

(3) 事务时间时态关系。其中关系元组具有相应事务时间标签,由子句 AS TRANSACTION 标识。

(4) 双时态关系类。由于其中包含有效时间,因此可分为“状态”和“事件”两种情形。

① 双时态状态关系:关系元组同时具有事务时间和有效时间标签,其中有效时间通常为时间期间集合,由子句 AS VALID [STATE] AND TRANSACTION 标识。

② 双时态事件关系:关系元组同时具有事务时间和有效时间标签,其中有效时间通常为表示相应事件发生的时间点集合,由子句 AS VALID EVENT AND TRANSATION 标识。

【例 6-21】 创建双时态状态关系“处方”如下。

```
CREATE TABLE 处方(
    病人姓名 CHAR(10),
    医生姓名 CHAR(10),
    药名 CHAR(30),
    剂量 CHAR(30),
    服药间隔 INTERVAL MINUTE),
AS VALID [STATE] DAY AND TRANSACTION;
```

说明: 语句中属性“服药间隔”表示每隔多少分钟服药一次。以 AS 引领的子句表明所建立的是双时态状态关系,具有效时间和事务时间两个时态属性。有效时间由时间期间表示且粒度为天;事务时间粒度由系统决定。

6.6.2 双时态关系数据查询

时态关系操作需要具有封闭性,即时态关系元组经过数据操作(查询和更新等)结果还应当具有时态元组的结构形式,即操作前具有怎样的时间标签,操作后还应当具有相应形式的时间标签。由于关系操作(如查询中的投影和连接)会使得关系元组发生改变,因此时态关系操作的一个基本点就是如何为改变后的关系元组赋予适当的时间标签。

从形式上来说,时态关系查询可分为隐式具有时间约束和显式带有时间约束两种情形。

1. 不显式带有时间约束查询

当查询要求中不显式具有时间约束时,由于时态关系运算封闭性要求,查询处理结果应该还是具有相应时间标签的时态元组集合。但对于用户而言,可以根据需要确定在最终输出结果中是带时间标签的时态关系或不带有时间标签的常规关系,而这可以通过在查询关键词 SELECT 之后添加关键词 SNAPSHOT 与否实现。

【例 6-22】 查询所有服过红霉素药物的病人姓名。

```
SELECT SNAPSHOT 病人姓名
  FORM 处方
  WHERE 药名 = '红霉素';
```

说明: 查询中无时间要求且无须时态标签,通过添加 SNAPSHOT 就可以使得在查询结果去除相应时间标签而只输出病人姓名列表。

【例 6-23】 查询曾服过红霉素的病人姓名及其服药期间。

```
SELECT 病人姓名
  FROM 处方
  WHERE 药名 = '红霉素';
```

说明: 查询结果为所有服过红霉素的病人姓名及其服药期间的集合。如果病人连续服用此药,集合有若干服药期间首尾相连,则 TSQL2 将这些服药期间自动归并为一个更大时间区间。

(1) 时态关系投影。设有时态关系模式 TR(A_1, A_2, …, A_k, VTs, VTe, TTs, TTe),其中 A_1, …, A_k 为 TR 常规关系属性。如需对 TR 在属性 A_{i1}, A_{i2}, …, A_{ip} ($ip<k$) 上投影,此时查询语句为:

```
SELECT Ai1, Ai2, … , Aip
  FROM TR;
```

查询结果为一个有改变了的元组构成的新关系 TR_0(A_{i1}, A_{i2}, …, A_{ip})。由于投影后可能产生常规属性完全相同而仅时间标签不同的元组,如前所述,此时需要按照 TSQL2 基本要求,需将其中时间标签进行归并。当其中多个有效时间期间"首尾"相接,就将它们归并成一个时间区间,从而得到一个新的时间期间集合作为相应元组的时间标签。此时,由于关系元组和相应时间标签都发生了变化,这就相当于在查询过程中或过程后进行相应的时态关系"重构"(restruction)。这种由投影引发构造新时态关系的过程就是时态关系重构,其要点是在 FROM 等子句中使用投影和重构之后的新的关系。正是由于能够在查询子句中重构新的时态关系,TSQL2 就可以通过时态关系重构机制而方便地获得所需时间标签。

【例 6-24】 查询凡服药必有红霉素的病人姓名。

```
SELECT SNAPSHOT P1.病人姓名
  FROM 处方(病人姓名) AS P1,处方(药品)AS P2
  WHERE P2.药名 = '红霉素'
    AND VALID (P2) = VALID(P1);
```

说明: 在 SELECT 子句中使用关键字 SNAPSHOT,故查询结果为病人姓名列表。

在 FROM 子句中使用了重构关系 P1 和 P2。其中,P1 是"处方"在"病人姓名"上投

影，时间标签是病人所有服药期间集合，而无论处方是那个医生所开、是何种药品及剂量和服药间隔如何。P2 相当于“SELECT 病人姓名，药名 FROM 处方”。此时可能就需要进行“重构”。在应用条件 VALID（P2）＝VALID（P1）时，隐含了条件 P1. 病人姓名＝P2. 病人姓名。P2 的时间标签是指病人服某种药的期间集合。在 WHERE 子句中，存在 P2 药名为“红霉素”的限制条件，因而 VALID（P2）是指病人服用“红霉素”的期间集合，而 VALID（P1）是病人服用任意药的集合。VALID（P2）＝VALID(P1)，则说明该病人凡是服药必有“红霉素”。

（2）时态关系连接。在时态关系连接过程中，除了常规连接条件之外，从构建连接后元组有效时间出发，还需要确定连接后新元组的时间标签，这就需要分析处理被连接两个元组时间标签的关系问题。

【例 6-25】 查询与红霉素同时服用其他药品名、服用此药病人姓名及同时服用的期间。

```
SELECT P1.病人姓名,P2.药名
  FROM 处方 AS P1,处方 AS P2
  WHERE P1.药名 = '红霉素'AND P2.药名<>'红霉素'
                      AND P1.病人姓名 = P2.病人姓名;
```

说明：FROM 子句中为关系“处方”定义了两个元组变量名，即 P1 和 P2。这些元组变量名称为关联名。对于关联名 P1 和 P2，除须满足 WHERE 子句要求外，还须满足默认条件：P1 和 P2 所代表元组在有效时间上相交，查询结果中的有效时间是相应相交部分。

（3）时态关系划分。在 TSQL2 中，即使元组初始的时间标签都为时间期间或时间点，但如果是先进行投影之后再实行其他查询操作，就有可能出现投影后元组时间标签为时间元素的情形。此时关系元组可表示为：

$$(a_1, a_2, \cdots, a_n), \{p_1, p_2, \cdots, p_m\} \tag{6.1}$$

其中，$(a_1, a_2, \cdots, a_n)$为非时态属性；$\{p_1, p_2, \cdots, p_m\}$为时间标签，而 $p_1, p_2, \cdots, p_m$ 为时间期间。在实际应用中，可能会出现需要“一个一个”甄别其中时间标签的情形，因此从技术上考虑，需要将式(6.1)转换为式(6.2)的形式：

$$\begin{array}{ll} (a_1, a_2, \cdots, a_n), & p_1 \\ (a_1, a_2, \cdots, a_n), & p_2 \\ \cdots & \cdots \\ (a_1, a_2, \cdots, a_n), & p_m \end{array} \tag{6.2}$$

即将一个元组变为 m 个元组。这种变换机制称为时态关系划分(partition)。当然这种时态关系划分不符合 TSQL2 中基本要求，出现多个除时间标签外其余部分都相同的元组，因此不能出现在输入的关系表和输出的关系表当中，但可以出现在“中间”处理过程当中。因此，TSQL2 允许此种划分出现在 FROM 子句。TSQL2 实际应用中，如需将所涉及关系进行时态划分，需在其后添加关键词(PERIOD)。时态关系划分通常用于具有连续时间条件的查询。

【例 6-26】 查询连续服用同一种药超过 6 个月的病人姓名、药名和服药期间。

```
SELECT SNAPSHOT 病人姓名,药名,VALID(P)
  FROM 处方(病人姓名,药名) (PERIOD) AS P
  WHERE CAST(VALID (P) AS INTERVAL MONTH)
       > INTERVAL'6'MONTH;
```

说明：处方关系先对病人姓名和药名两个属性进行前述时态重构，得到"中间"时态关系为形如(病人姓名，药名)二元组的集合，而其中每个二元组就有可能具有形如式(6.1)的时间标签。此时需要对对重构后关系进行时态划分，通过关键词(PERIOD)得到关系 P，此时 P 中每个元组时间标签为单一时间期间而非时态元素的集合，系统使用此时间标签判断是否连续服某药超过 6 个月。此外，CAST 是把期间转换为以月为单位的间隔的转换函数。SELECT 子句中有关键字 SNAPSHOT，查询结果中无时间标签。为表示连续服药实际时间，语句中以 VALID(P)作为查询结果的第三个属性。查询结果中同一病人姓名和药名可能对应多个元组，每个元组有不同期间。例如，在 SELECT 子句中不用保留字 SNAPSHOT，也可得到类似的结果。

2. 具有时间约束查询

显式具有时间约束查询可以分为具有效时间、事务时间情形。

1) 具有效时间约束查询

实际应用中使用 VALID 子句限制查询结果中有效期间的范围。

【例 6-27】 查询在 2016 年，医生给 John 开过哪些药品。

```
SELECT 药名 VALID INTERSECT(VALID(处方),PERIORD'[2016 - 1 - 1, 2016 - 12 - 31]')
  FROM 处方
  WHERE 病人姓名 == 'John';
```

说明：查询结果是 John 在 2016 年曾经服过的药名，每个药名后附有一个时间标签，它是 John 在 2014 年服用此药期间集合。如果无 VALID 子句，则查询结果将是 John 曾服过所有药名，每种药名所附时间标签是 John 服用此药期间集合。

2) 具事务时间约束查询

以上查询未出现对事务时间的查询，而实际上双时态数据库中事务时间也是查询条件之一，但通常默认的都是查询数据的最新版本。设 T 为关系的元组，这相当于以下列条件作为查询事务时间的缺省条件：TRANSACTION(T) OVERLAPS(CONTAINS) CURRENT-TIMESTAMP。

当查询数据以往版本时，就需要明确地指明事务时间。

【例 6-28】 缺省事务时间查询：查询 John 的历次处方。

```
SELECT *
  FROM 处方
  WHERE 病人姓名 = 'John';
```

说明：查询未显式指明事务时间条件，因而按当前事务时间查询。如果处方被修改过，则查询结果为最新修改过的处方版本。

【例 6-29】 指定事务时间查询：查询自 2016 年 10 月 1 日所看到 John 历次处方的版本。

```
SELECT *
  FROM 处方 AS P
  WHERE 病人姓名 = 'John'
       AND TRANSACTION(P) OVERLAPS DATE'2016 - 10 - 01'
```

【例 6-30】 查询 John 在 2016 年 3 月 2 日使用过处方最近一次被修改的时间。

```
SELECT SNAPSHOT BEGIN(TRANSACTION(P2))
  FROM 处方 AS P1.P2
  WHERE P1.病人姓名 = 'John'AND P2.病人姓名 = 'John'
        AND VALID(P1) OVERLAPS DATE'2016 - 03 - 02'
        AND VALID(P2) OVERLAPS DATE,2016 - 03 - 02'
        AND TRANSACTION(P1) MEETS TRANSATION(P2)
```

说明：P1、P2 是两个关联名，分别代表处方中两个元组，其中病人姓名都是 John。有效期间都覆盖 2016-03-02，都是 John 在 2016 年 3 月 2 日使用过的处方。但 P1 与 P2 事务时间不同。P2 事务时间紧接在 P1 之后，因此，P1 是未修改过的处方，P2 是修改后的处方，Pl 修改时间在 TRANSATION(P2)开始之前。如果该处方仅修改一次，则此时间即 BEGIN (TRANSACTION(P2))就为所要求的答案；如果该处方多次修改，则查询结果将是该处方历次被修改的时间，而最后一次的修改时间就是所要求的答案。

6.6.3 双时态关系数据更新

TSQL2 数据更新包括数据插入、删除和修改。SQL 中的相应更新语句都可推广到 TSQL2。

1. 数据插入

插入数据时，如果关系"处方"已有属性相同的元组，则新插入数据应与该元组归并，即在相应元组时间标签中添加插入数据有效期间。只有当相应关系中没有任何元组的属性值与插入元组完全相同时，插入的处方元组才能作为一个单独元组插入到关系"处方"中。

【例 6-31】 插入一处方元组，但处方元组有效时间终止时刻待定。

```
INSERT INTO 处方
  VALUES('John','White','Vitamin E','100mg' INTERVAL '8:00' MINUTE);
```

说明：服药间隔数据类型为 INTERVAL，值为 8，时间粒度为分。在本例中未指明时间标签，则在插入时时间标签取缺省值：VALID PERIOD (CURRENT-TIMESTAMP, NOBIND(CURRENT-STAMP))。

CURRENT-TIMESTAMP 表示指插入时的当前时间，其粒度由系统决定。括号中第一、二项分别表示有效时间的开始和终止时刻。例如，插入时终止时刻值明确就直接填入，若终止时刻在开处方时难以确定，需根据药效确定，TSQL2 使用变量 NOBIND (CURRENT-STAMP)予以记录，查询时该变量就与当前查询时间绑定。注意，这种变量形式在 SQL 中不被允许。

【例 6-32】 插入一处方，处方有效终止时刻确定。

```
INSERT INTO 处方
  VALUES('John','White',Vitamin E','100mg'INTERVAL'8:00'MINUTE)
    VALID PERIOD'[2017 - 02 - 01,2017 - 08 - 31]'
```

说明：插入的这条处方的有效期间为 2017 年 2 月到 8 月。

2. 数据删除

【例 6-33】 删除 2017 年 10 月开给 John 的所有处方。

```
DELETE FROM 处方
  WHERE 病人姓名 = 'John'
  VALID PERIOD'[2017 - 10 - 01,2017 - 10 - 31]'
```

说明：凡在 2017 年 10 月开给 John 所有处方都将被删除。如果有处方有效期只部分在 2017 年 10 月，则从有效期中删除 2017 年 10 月中部分而保留其余部分。

3. 数据修改

【例 6-34】 改变 John 在 2017 年 2 月至 8 月的 Vitamin E 的剂量为 50mg。

```
UPDATE 处方
  SET 剂量 TO'50mg'
  VALID PERIOD '[2017 - 02 - 01,2017 - 08 - 31]'
  WHERE 病人姓名 = 'John'AND 药名 =  Vitamin E
```

本 章 小 结

具有时间信息的数据是一类特殊的数据，但其讨论和研究还是遵循一般数据的技术路线，即需要讨论相应的数据结构、建立适于计算机表示和存储的数据类型及基于相应类型的数据运算。在 TDB 中，采用基于线性的离散数据结构；设计了时间点、时间期间和时间元素及时间戳的时间类型；建立了跨类型的时间数据代数运算和相应的关系演算，从而为研究开发 TDB 相关技术奠定了必要的原理和技术基础。

在 TDB 技术实现方面不能笼统地考虑时间，需要根据数据库管理时间的自身特征对时间进行不同角度的分析与探讨，基本的问题角度在技术上通常称为维度，这就引入了基于时态数据管理的 3 个基本维度，即用户定义时间、有效时间和事务时间。引入时间维度的意义在于能够对时态数据库进行基本的分类，即分为基于用户定义时间的快照数据库、基于有效时间的历史数据库、基于事务时间的回滚数据库和同时基于有效时间与事务时间的双时态数据库。

时态数据模型是构建时态数据库的基础。基于历史关系数据模型 HRDM 可以构建起历史数据库，这可以看作是一个非 1NF 的时态 RDB；基于双时态数据模型 BCDM 可以构建双时态数据库，如果采用其 RDM 形式，就可看作一个 1NF 的时态 RDB。

数据的有效时间可以是不断延伸的，在实际问题当中，常常会出现数据有效时间在对该数据进行操作时尚未结束，但何时结束又难以完全确定的情形。时态数据库处理此种情形的有效途径就是引入有效时间变量 Now。数据事务时间也有类似情形，需要引入事务时间变量 uc。

有效时间变量 Now 在数据操作时绑定于操作对应的当前时间 CT。但是由于事务时间与有效时间的非同步性质，Now 并不总表示当前时间，还可以表示过去时间和将来时间，这就在时态数据库中引入了 Now 语义的分析处理及查询结果不确定性问题。带有变量的双时态数据查询与更新操作时需要对 Now 语义进行比较精细的分析和处理。

TSQL2 是对应于 BCDM 的双时态 RDB 语言，其基本点是不能出现只有时间属性不同的时态元组，因此，将相应时间属性进行必要的处理，如时间归并就是时态数据操作的一项基本任务。作为 SQL 的时态扩展，TSQL2 是时态数据操作的标准规范。

RDB 可以看作是数据管理技术在事务维度上的展开，而 SDB 和 TDB 可以看作是在事务维度基础上分别添加上了空间维度和时间维度。通过日常经验可知，当对所涉及问题进行空间和时间因素考量时，实际上就到达了一个对问题更为深入和更加全面精细的研讨阶段。因此，SDB 和 TDB 出现应该是数据库技术发展的一种必然，是与整个计算机科学技术发展和应用领域拓广相适应的。

主要参考文献

[1] 汤庸. 时态数据库导论[M]. 北京：北京大学出版社，2004.

[2] 何新贵，唐常杰，李霖，等. 特种数据库技术[M]. 北京：科学出版社，2000.

[3] Allen J F. Maintaining Knowledge about Temporal Intervals[J]. Communications of the ACM，1983，26(11)：832-843.

[4] Yong Tang，Xiaoping Ye，Na Tang. Temporal Information Processing Technology and Its Application [M]. Beijing：Tsinghua University Press and Springer，2010.

[5] 叶小平. 基于时态变量对象关系模型及代数运算[J]. 计算机研究与发展，2007，44(11)：1971-1979.

[6] 叶小平，汤庸. 时态变量“Now”语义及相应时态关系运算[J]. 软件学报，2005，16(7)：838-845.

第7章 XML 数据库

XML 是 eXtensible Markup Language(可扩展标记语言)的缩写。标记语言的功能是对文档提供规范化的描述,用以指明文档中哪些部分是具有实际语义的内容,哪些是用以描述这些内容的标记,同时指出这些标记应当如何使用。当前,XML 已经成为互联网数据表示与交换的标准。XML 的要点在于能够对文档文件的数据内容即所表达语义进行适当描述,使得计算机系统能够通过相应标记对数据内容进行识别和对数据文件实施有效管理,由此可知,XML 技术和数据库技术的结合形成 XML 数据库也就是顺理成章的了。本章将简述 XML 文档、XML 数据模型和 XML 数据索引等 XML 数据库的基本技术。

7.1 XML 文档

作为一种使用方便灵活的元语言,XML 文档具有良好的数据存储格式、应用可扩展性和便于网络传输等特性,这使得 XML 适用于网络数据自身的信息提取和数据之间的信息交互。自 1996 年推出后,XML 得到迅速发展和广泛应用。如今,XML 已被看作是计算机科学技术发展史上具里程碑意义的重要技术之一,并将成为一种对现实技术世界产生重要影响的规范标准。

7.1.1 标记与标记语言

标记是对文档内容做出说明但不需要实际输出的字符集,如高级程序设计语言中的关键字和注释行都可以看作是某种意义下的符号标记。作为一种元语言,XML 相对于表达"数据"内容的常规语言具有更高的语义级别和逻辑层次。

1. 标记语言:元语言

(1) 标记(markup):文档中为了"解释"相关内容的含义但不必"实际"输出的一些注记字符。换言之,"标记"可看作一种表达文档中元数据信息即关于数据集本身构成信息的方法,其特征是将相应文档划分成各种部件并对其分别加以标识。

(2) 标记语言(markup language):对文档中的标记进行有效使用的一种规范性和形式化的描述系统,其作用在于提供标记使用的基本规范和语法约束。标记语言使用文本串或其他"标记符"来界定和描述文档中的数据内容。

按照使用的实际效果,标记语言可分为下述两种类型。

① 格式化标记语言:对于文档输出格式进行描述和标记的语言。例如,HTML(Hyper Text Markup Language)就是一种用于网页文档输出的格式化标记语言,它通过指定的标

签集合对网页所需要采用的输出格式进行描述和刻画。从文档输出角度来看,使用格式标记语言的意义在于允许同一内容的文档在不同情况下可根据实际需要而具有不同的输出格式。

② 功能性标记语言:对于文档语义内容进行描述和标记的语言。例如,本章所要讨论的 XML 就是一种具有语义描述功能的标记语言。从文档描述角度考虑,使用功能标记语言的意义在于明确标记文档中各个部分在语义上分别表示怎样的内容,这样就会有助于在查询过程中进行数据语义识别,由此得到文档查询结果并不是通常存储的整个文档,而是能够像 RDB 那样,从一个文档或多个文档中抽取相关部分数据进行整合而得到相应查询结果。

正是由于能够通过标记语言进行文档内容的语义识别,这就在本质上隐含了基于数据库管理 XML 文档的可能性。

引入标记语言的结果是将文档中的"内容"部分和"说明"部分进行适当分离。如果将文档内容看作是"硬"的实质性部分,将说明部分看作是"软"的逻辑性部分,那么在标记语言中,这种"内容"和"说明"的分离就可以类比于在数据管理技术演变过程中将 DBMS 从物理文件处理过程中分离出来。

在 XML 之前,除了已经实际使用的格式化标记语言 HTML 之外,实际还有早先提出的功能性标记语言(Standard Generalized Markup Language,SGML)。作为一种语义描述工具,SGML 自 20 世纪 80 年代已开始使用,它具有良好的扩展功能,在数据分类与数据索引过程中发挥着良好的作用。但 SGML 机制过于复杂且价格昂贵,难以有效满足大众化网络社会的实际需求。实际上,可以将 SGML 看作 XML 的一个超集。

2. 语义标记语言 XML

XML 可以看作是针对 SGML 和 HTML 的弱项和不足方面进行了"补充"或"修订",同时对两者的优势进行了"扩展"和"整合"。W3C 于 1998 年 2 月提出了推荐标准 XML 1.0 (http://www.w3c .org/TR/R EC-xml)。为正确使用 XML 文档, W3C 又为 XML 标准化定义了一套应用程序编程接口(Application Programming Interface, API),同时,还设计了一套特殊的基于事件的替代 API。与 HTML 一样,XML 基于 ISO/IEC 10646 字符集标准(等同于 Unicode 标准)中定义的通用字符集(Universal Character Set,UCS)。

XML 继承了 HTML 的特性,但并非 HTML 的直接替代。

(1) HTML 注重数据及表达方式,XML 注重数据本身内在的语义。

(2) HTML 使用固定的标记集合,而 XML 没有指定的标签集合而允许用户根据应用需要设计和选择相应的标签集。

XML 文档样式表现技术主要由数据驱动通过另一个被称为样式单的文档来实现,其中,设计者会格式化样式和决定何时应用样式的规则。XML 样式单可用于多个文档,以产生类似的样式效果。样式和规则在显示时会应用到 XML 的数据上,并可转换为 HTML 或其他数据格式。

3. 语义识别与信息集成

XML 最初并不是作为一项数据库技术进行设计开发,其实际应用驱动是对于网络文本文档的管理。实际上,当网络上两个应用程序需要进行通信或需要从多个应用程序中整合信息时,相关数据的语义识别和信息集成就成为网络数据管理的迫切需求。此时,使用

XML 格式表示网络文档文件，其具有语义表示功能的标签就能发挥重要的作用。而依据语义对存储数据进行实际查询正是数据库管理数据的核心课题，因此在 XML 使用过程中，必然会涉及许多的数据库问题，从而也就自然而然地进入到 XML 数据库技术的范畴。

1）语义识别

【例 7-1】 设有如下文档信息："John，White，XML database，＄32，Springer Valag Berlin"。这样表示的文档字符串对于"人"而言，其表达内容也就是语义信息是清晰而明确的，但将其存储在计算机中之后，就只能是一个"字符串"，此时人们并不能有效地通过使用计算机来识别和提取出存储其内的语义信息，如"author"是"谁"和"publish house"是"哪一家"等。但如果对上述文档添加适当标记组织成下述 XML 文档形式。

```
<?xml version = "1.0" encoding = " ISO - 8859 - 1" standalone = "yes"?>
< book year = "2017">
    < bookname > XML Datebase </bookname >
    < author >
        < firstauthorname > John </firstauthorname >
        < secondauthorname > White </secondauthorname >
    </author >
    < price > $ 32 </price >
    < publish house > Springer Valag Berlin </ publish house >
</book >
```

此时，虽然计算机仍旧不能理解"author"和"publish house"的语义，但当人们通过输入"author"和"publish house"到计算机中进行查询时，计算机就能通过常规模式匹配技术迅速地找到标记对< author >…</author >和< publish house >…</publish house >并通过适当方式提取出标记对其中的内容"John，White"和"Springer Valag Berlin"予以输出。由此可见，使用标记语言实际上提供了一种使得计算机对其中的文档按照语法存储进行语义识别和信息提取的适当途径，就如同计算机对文档中的"语义"具有了一定程度的"认知"能力一样。这样就使得计算机所能够处理的数据信息范围得到极大扩张，"似乎"提高了计算机进行语义识别的"智能"本领。

在数据库技术学习中已经知道，数据的语义识别对于数据文件中的"个别"信息的提取极其重要。文件系统管理数据只能对存储的数据进行文件名的识别，数据存储的技术单元只能是数据文件本身，不能深入到数据文件内部；其查询也就只能是文件的"整体"查询，即在众多文档中通过适当方法搜索到相关文档，而对于所需要的信息内容只能在该文档中通过遍历进行查找，不能有效从文档中直接提取所需"个别"的信息。究其原因，就是由于一般数据文件没有相应的语义识别机制，同时也不具有能够描述和表达语义的数据逻辑结构。作为一种特殊的文档文件，XML 文档通过标签进行文档内容的语义识别标记，同时具有"嵌套"层次作为相应的逻辑数据结构，从而具有一般文档文件所不具有的语义识别功能，为 XML 文档的数据库管理提供了有效途径。

2）Web 信息集成

一般而言，Web 数据管理的核心是信息提取和数据集成。如上所述的"语义识别"就是"信息提取"的前提，而"信息提取"就是从诸如 XML 等文档据中归纳反映出当前状态的结构模式，以便在此基础上对半结构化数据进行查询、计算和优化，从而得到所需要的信息内

容。进行模式提取后的具有不同来源的 Web 信息彼此之间通常呈现出异构的情形,难以进行统一的管理操作,因此需要进行"数据集成"。实际上,如果能将 Web 数据进行有效集成,使得相应数据能够遵循统一的标准要求和规范格式,将会大大有利于 Web 数据的信息提取,同时也有利于基于数据库的 Web 数据管理。XML 就是当前 Web 数据交换和集成表示的基本标准。

基于 XML 的 Web 数据集成系统通常遵循 Wiederhole 集成体系结构。Wiederhole 集成体系分为 3 层。

(1) 数据包装层:用于完成数据源特定查询接口包装和接收系统发出的查询,并将其转换为数据源特定查询形式,执行后返还查询结果。

(2) 数据集成层:接受包装层面的数据输入和输出整个集成系统的数据模式,完成数据源模式和全局集成模式的相互映射。用户基于全局模式发出查询请求,集成层面将其转换为数据源特定查询格式,发送到各个数据源分别进行相应操作。当系统为提高查询效率而采用物化视图格式时,集成层面则完成相应物化视图管理。

(3) 数据查询层:用户(应用程序)根据全局集成视图发出查询要求,获取最终查询结果。

Wiederhole 集成体系如图 7-1 所示。

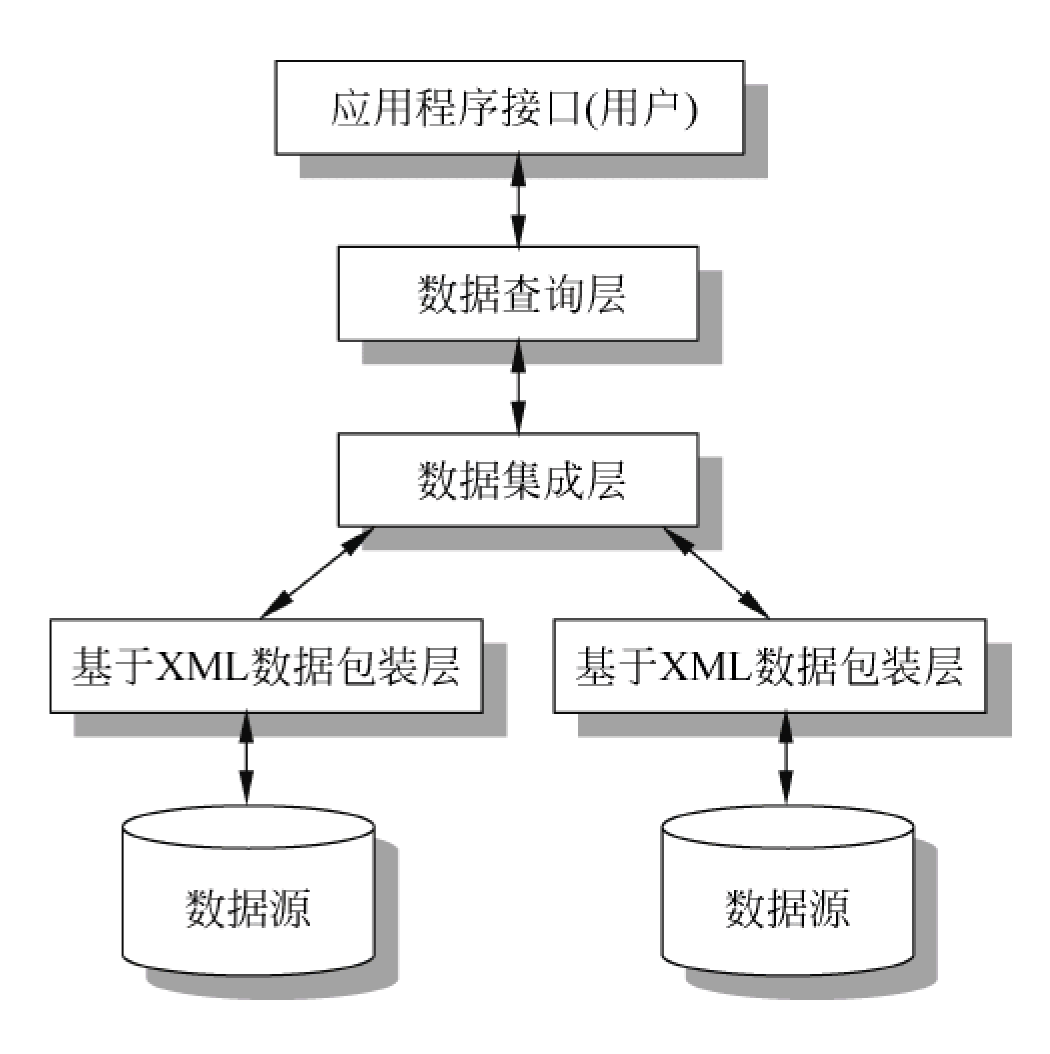


图 7-1　Wiederhole 集成体系

7.1.2　XML 文档组成与良好 XML 文档

XML 文档实际上是一个连续的字符流(stream of characters),字符流中出现的字符需要具有一定的顺序,通过不同的标记标识各个不同的语义模块并按照一定规范将它们组织起来。

1. XML 文档组成

XML 文档基本组成要素有"XML 文档声明""元素与属性""引用与注释"和"名空间"等。

1) XML 文档声明

在例 7-1 中,第一行表示 XML 声明(XML declaration),这是对文档处理环境和要求的

表述与解释,不涉及文档本身内容。

XML 说明放置在“<?”和“?>”之间,其中包含以下 3 个属性。

(1) version 属性:说明当前文档以 XML.1.0 标准编写,告之 XML 解析器文档使用的版本。

(2) encoding 属性:表示当前文档使用字符集 ISO-8859-1 进行编码。

(3) standalone 属性:表明当前文档是否引用其他文档内容,“yes”表示无引用,文档独立。

XML 标准规定,XML 文档必须冠以 XML 文档说明,但 version、encoding 和 standalone 当中只有 version 是必需的,其余两个属性是可选的。如果 3 个属性都使用,则三者顺序为 version 在前,encoding 其次,standalone 最后。

在例 7-1 中,第二行以后部分为文档实例,它构成 XML 的主体。

2) 元素

XML 文档中标记需要成对出现,即以开标签<标记名>开始,以闭标签</标记名>结束。每组成对的标记就构成 XML 中的一个元素(Element)。

元素是 XML 文档中基本的语法成分并具有如下语法格式。

```
<Element Name> 元素内容 </Element Name>
```

这里 Element Name 是标记,为用户定义的元素名称。< Element Name >为开标签,</Element Name >为闭标签,两者之间是元素内容。

一个 XML 文档需要有一个根元素(Root Element),其他元素和文档内容都包含在根元素起始标记和结束标记之间。

元素可以嵌套包含子元素,嵌套层数不受限制。最外层元素就是根元素。

在例 7-1 中,根元素是“book”,子元素分别为“bookname”,“price”和“publish house”,而“author”还分别有两个子元素“firstauthorname”和“secondauthorname”。

3) 属性

属性(attribute)表明相应元素的某方面特征,其中包括属性名和属性值。属性名是一个用户自定义的标记,属性值用引号括起来,单双引号均可,但必须配对一致。属性名和属性值以等号相连接。在 XML 文档中,属性都放置在相应元素的开标签中,如例 7-1 中的 gear。

在 XML 数据表示角度,引入属性有以下两方面的意义。

(1) 一般属性体现半结构数据特征:属性将数据(元素)的取值与元素的标签联系起来,而元素标签的组织体现着数据文件的结构,因此,属性也可以看作是结构模式与数据内容结合的一个基本实现环节,即体现了 XML 文档的半结构数据特征。

(2) ID 属性实现 XML 和半结构数据对应:对于 XML 文档中的每一个元素,可以赋予数值 ID,用于唯一标识该元素开标签和闭标签所包含的段。数值 ID 就可以看作是该元素的基本属性值,而从半结构化数据观点来看,ID 就为每个结点提供了唯一名称,因此,通过 ID 属性,一个 XML 文档就可以看作一个半结构化数据(文件)

在某些情况下,子元素可以用属性表示,如例 7-1 中的 author 和 price 可以改用元素“book”的属性表示,其文档实例部分如下。

```
<book author = "John and White"Price = " $ 32">
```

```
    <bookame> XML database </bookame>
    <publish house> Springer Valag Berlin</ publish house>
</book>
```

注意，不是任何情况下都可用属性代替子元素，因为属性与子元素毕竟具有区别。

在同一元素中，属性不能重名，但子元素却可重名；属性在书写上有先后之分，但次序无关紧要，但子元素需要按照其书写次序排序。

一般而言，在元素与属性都能表示的场合，可有下述参考。

① 元素组成部分宜用子元素表示。

② 元素性质内容宜用属性表示。

③ 简短内容宜用属性表示。

④ 嵌套或较长内容宜用子元素表示。

元素可以没有内容只有属性，这样的元素称为空元素(empty element)。

4）引用

在编写 XML 文档时，经常需要进行引用(reference)本文档或其他文档的内容，为了减少工作量，可以将需要重复引用的内容定义为 Entity。Entity 的语法格式如下。

```
<!Entity 引用名"引用内容">
```

<!…>是 XML 的定义语句格式；<!Entity…>是一条定义 Entity 的语句；引用名(Entity Name)是引用内容的名称。在引用时，只须在引用处输入"& 引用名称；"即可。引用内容可以包括 HTML、文档、图形、声音和影视等任意情形，但如果含有 XML 文档中特定内容的符号，如<、>、& 等，就需要加以区分。为此，引用内容可以包装为：

```
<![CDATA[ 引用内容 ]]>
```

其中"<![CDATA["可以看作是开标记，"]]>"可以看作是闭标记，这两个标记表示对其中的内容增加了新的语法内涵，不做语法分析而原样引用。如此处理的 Entity 定义可以表示为：

```
<!Entity 引用名"<![CDATA[引用内容]]>">
```

5）注释

XML 文档在需要处添加上注释(comments)。注释的格式如下。

```
<! -- 注释 -- >
```

注释仅供人们阅读，机器处理时予以忽略。注释可以是任意内容，但不能带有"--"字符串。一个带有注释的 XML 文档如下。

```
<?xml version = "1.0" encoding = " ISO - 8859 - 1" standalone = "yes"?>
<! -- John 的个人资料 -- >
<person ID = "F44010219760708453">
    <name> John </name>
    <sex> male </sex>
    <birthday> 1976 - 07 - 08 </birthday>
    <phone>(020)56892379 </phone>
    <occupation> student </occupation>
```

```
<! —The markup is important because it records the structures of the document -->
</person>
```

6）名空间

与 HTML 仅作为文档标记语言不同，XML 是一种文档描述语言，各类应用都可定义在本领域中具有特定含义的专业标记，这些标记相当于该领域中的基本词汇表，这在 XML 中称为该领域的名空间(Name Space，NS)。由于一个领域可以有自身子领域，因此名空间中还可以有相应的子名空间，子名空间继承其上层名空间的标记，如同 C++中子类继承基类一样，当标记含义出现冲突时以子名空间定义为准。名空间可以通过 URI(Uniform Resource Identifier，统一资源标识符)在互联网上公布。通常，XML 文档的前部可标明所使用名空间的网址，用户可根据需要在实际应用中进行查阅。由于 URI 在网络中唯一，使用名空间机制的 XML 中标记含义也就唯一。

名空间一般定义格式为：

```
<标记名 xmlns: 名空间名称 = "URI">
```

其中，“标记名”是用户在文档中定义的标记；“xmlns”是 W3C 提供的最常用的公共标记名空间，其他各个名空间都是其子名空间，可以看作子名空间的固定前缀；“名空间名称”是用户为名空间定义的在文档中具有唯一性的名称；“URI”是名空间的唯一标识符。“xmlns：名空间名称＝"URI"”表示该用户定义的名空间是 xmlns 的子名空间。在不注明名空间情况下，通常表示名空间默认为 xmlns。

【例 7-2】 在例 7-1 中，如需要限定 book 元素及子元素命名作用范围，可由下述语句实现。

```
<?xml version = "1.0" encoding = " ISO - 8859 - 1" standalone = "yes"?>
< book year = "2017"xmlns: Sun Yat - Sen University  = "http://www.sysu.edu.cn">
  < SunYat - Sen University book name > XML Datebase </SunYat - Sen University book name >
  < Sun Yat - Sen University author >
       < first author name > John </first author name >
       < second author name > John </second author name >
  </Sun Yat - Sen University author >
  < Sun Yat - Sen University price > $ 32 </ Sun Yat - Sen University price >
  < Sun Yat - Sen University publish house > Springer Valag Berlin </ Sun Yat - Sen University
     publish house >
</book >
```

说明：在例 7-2 中，名空间“Sun Yet-Sen University”绑定于 URI“http://www.sysu.edu.cn”；bookstore 子元素 book 也被“Sun Yet-Sen University”所限制。需要指出，在 XML 中，如果子元素没有使用名空间，则默认接受其父元素名空间的约束。

2. 良好 XML 文档

如果一个 XML 文档满足如下约束条件，则称其为良好(Well-Formed)的 XML。

1）标记基本类型

XML 文档中使用了以下 5 种标记类型。

(1) 元素(element)标记。

(2) 属性(attribute)标记。

(3) 注释(comment)标记。

(4) 处理指令(Processing Instruction,PI)标记。

(5) 实体(entity)标记。

2) 标记语法约束

XML 文档中标记使用符合以下语法约束。

(1) 所有 XML 标记需要合理嵌套。

(2) 所有 XML 标记需要有一个相应结束符号。

(3) 所有 XML 标记需要区分大小写。

(4) 所有 XML 标记的属性必须用""引起来。

显然,bookstore.xml 是一个良好的 XML 文档。

3) 标记命名规则

XML 文档中标记名称遵循以下命名规则。

(1) 名称中可以包含字母、数字及其他字母。

(2) 名称不能以数字或"_"(下画线)开头。

(3) 名称中不能包含空格。

(4) 名称不能以字母 xml (或 XML 或 Xml ..)开头。

按照上述要求,例 7-1 和例 7-2 都是一个良好的 XML 文档。

7.1.3 DTD 与有效 XML 文档

XML 允许用户自行定义所需要的标记。在实际应用中,对于同一个文件内容,不同编程人员可能写出不同的 XML 文档。如果无事先的协议来约束,基于计算机的 XML 文档自动交换和处理很难进行。为此,需要设计出统一格式的 XML 文件,这个统一格式就是文档类型定义(Document Type Definition,DTD)。文档类型定义通过一系列彼此相关的声明来描述文档结构模式。这些声明包括元素名称、元素出现顺序、元素数据类型、元素出现次数、可选择的元素、元素的属性、注释及实体声明等。

如果 XML 文档出现差错,对应的网页就可能不被显示,因此需要制定和达成一定的协议,对 XML 实行严格的语法解析。XML 解析器就是检验一个文档是否符合 XML 语法规则的处理工具,在解析过程中,解析器将会报告 XML 文档可能会出现的各类错误。DTD 是一种保证 XML 文档格式正确的有效方法,通过比较 XML 文档和 DTD 文件就可检查文档是否符合规范,元素和标记使用是否正确。

1. DTD 概念

一个 DTD 文件通常包含元素定义规则、元素之间关系定义规则、元素可使用属性、实体和符号规则。DTD 是一个 ASCII 的文本文件,后缀名为.dtd,如 hello.dtd。使用 DTD 的意义在于不同语义内容的 XML 文档能够实现结构和格式的共享。DTD 和满足 DTD 要求的 XML 文档的关系类似于数据模式和数据实例的关系。

目前,已经有数量众多的编写完备的 DTD 文件可供利用。针对不同的行业和应用,这些 DTD 文件已经建立了通用的元素和标签规则,在应用过程中用户并不需要自己重新创建,只要在已有 DTD 基础上加入所需新标识即可。实际应用中,用户也可创建自己的 DTD,使得相应文档配合更加完善适用。建立用户所需 DTD 通常一般只要定义 4～5 个元

素集合。DTD具有如下特征。

(1) 与数据模式类似,DTD实际上定义和限制XML文档中数据模式,即在一个元素中是否存在子元素和属性。

(2) 与数据模式不同,DTD本质上并不限制XML文档中数据类型,具有相当的灵活性。

在上述观点下,DTD可看作是定义一个元素中子元素类型的清单,即是一种在结构化与半结构之间取得某种平衡的结构模式。

2. DTD语法

DTD一般定义格式如下。

```
<!DOCTYPE 根标记[各元素定义,各元素属性定义,… ]>
```

其中,<!DOCTYPE…>表示其内容是文档类型定义,其后面方括号中为定义的内容。需要注意,在XML中,用逗号分开的项目有序,而用空格分开的项目无序,元素定义应当在属性定义之前,属性后面还可有其他定义,如引用定义等。所有定义都放在方括号内,并冠以根标记。

1) DTD中元素定义

在DTD中,可以通过关键词标记ELEMENT来声明元素,定义格式为:

```
<!ELEMENT 元素名称 元素内容描述>
```

按照其所包含内容,元素通常分为下述四类。

(1) 空元素类型。定义格式为:

```
<!ELEMENT 元素名称 EMPTY>
```

这种类型在XML中使用空元素标记,元素中没有内容。

(2) ANY元素类型。定义格式为:

```
<!ELEMENT 元素名称 ANY>
```

这种类型在XML中可以包含任意内容,但通常只将XML中根元素定义为此种类型。

(3) 父元素类型。定义格式为:

```
<!ELEMENT 元素名称 (|子元素名称 1|子元素名称 2| … … )>
```

此元素类型的特征是元素中可以包含子元素。

(4) 子元素符号。在DTD中通过正则表达式规定子元素出现的顺序和出现的次数。在DTD中正则表达式中,子元素"顺序"和"次数"由下述符号表示。

① 子元素顺序符号:分为子元素集合和子元素序列两种情形。

子元素集合,特点是所涉及的子元素名称连写:子元素名称1 子元素名称2 …… 子元素名称 n 。此时不需要遵照任何顺序要求。

子元素序列,特点是所涉及子元素名称按照先后顺序用逗号分隔:子元素名称1,子元素名称2,……,子元素名称 n。此时表示"并(AND)"语义,需要严格遵循顺序要求。

② 子元素次数符号:如果同名数据对象(如同名子元素)需要出现多次,则可在相应数据对象标记的右上角或后面添加 *(表示0次或多次)、+(表示1次或多次)或?(表示0次

或1次)三类符号。

如果数据对象右上角或后面无符号则表示取且取一次。

(5) 混合元素类型。混合元素类型定义格式为:

```
<!ELEMENT 元素名称 (#PCDATA|子元素名称1|子元素名称2|……)>
```

这种类型的特征是元素中可以同时包含文本内容和子元素,但其中文本内容必须是#PCDATA类型,即是可以进行解析的字符文本,不能在其中拥有自己的子元素。

2) DTD中属性定义

在DTD中,属性定义格式为:

```
<!ATTLIST 所属元素名称{属性名称 属性值类型 属性可选性}>
```

其中"{.}"表示一个元素可以定义多个属性。在多个属性情形下,通常是每个属性定义单独占据一行以增加可读性。

在DTD属性定义中,根据对属性取值与否或怎样取值,可以将属性类型分为以下4种。

(1) #DEFAULT value:表示属性值有默认值value。

(2) #REQUIRED:表示属性值是必需的,若缺少属性值,则取默认值(default value),而默认值无须在定义中标明。

(3) #IMPLIED:表示属性值是可选的,即可以选用,也可以不选用。

(4) #FIXED:表示该属性值是必需的,若缺少属性值,则取其默认值,但默认值必须在定义中标明。

3) DTD数据类型

在XML中,元素内容和属性的取值都是文本形式,此时需要考虑相应的数据类型。在DTD中,元素内容和属性取值都看作是字符串,字符串是XML中最基本的数据类型。

按照是否需要解析即语法分析,通常将这些字符串类型分为PCDATA类型和CDATA类型两种。因此,DTD中文本定义主要是对文本PCDATA类型和CDATA类型的定义。

(1) PCDATA。PCDATA(Parsed Character DATA)这种数据类型在前面已经多次提及,其特征是需要通过解析器的语法分析,其中字符串不能含有XML中有特定意义的符号,或者对这些符号进行了替换。

(2) CDATA类型。CDATA(Character DATA)类型中字符串可能含有XML中具有特定意义的字符串,其特征是经不起语法分析。因此CDATA类型的数据不需要通过解析器的语法分析。

(3) 其他类型。按照数值取舍或其他技术角度考虑,除了前述的DEFAULT value、REQUIRED、IMPLIED和FIXED之外,DTD中还有如下基本数据类型。

① ID:识别元素的标识符。类型为ID类型的属性提供了所属元素唯一的标识,出现在一个元素中的ID一定不能出现在另一个元素中,同时,一个元素只能有一个属性具有ID类型。

② IDREF(IDREFS):属性值是其他元素或本元素的ID。由此可见,类型为IDREF的属性是对另一个元素的引用,该属性必须包含一个文档中某个元素的ID类型属性中出现过的值。IDREF类型的属性值可以是多值的,此时,用关键词IDREFS标识,同时值与值之间用空格分开。

ID 和 IDREF 类型在面向对象数据库和对象关系数据库中扮演中同样的引用机制。

① ENTITY(ENTITIES)：属性值是引用名，可以是多值，值与值之间以空格分开。

② 枚举类型：在所列举的属性值中选取其一。

需要说明的是，在 DTD 中，在表示元素或属性数据取值类型时，通常需要将 # 置于 PCDATA 或 CDATA 之前，以避免与子元素名或属性名混淆。

3. 有效 XML 文档

DTD 文件可通过直接调用包含在 XML 文档中的 DTD 文件和调用独立存在的 DTD 文件。良好 XML 文档如果符合其相应存在的 DTD 规范要求，就称其为有效的(valid) XML 文档。例 7-1 中 XML 文档 bookstore. xml 对应的 DTD 文件如下。

```
<!ELEMENT book(bookname,author * ,price)>
<!ATTLIST book year #CDATA,REQUIRED>
<!ELEMENT bookname(#PCDATA)>
<!ELEMENT author(firstauthorname,secondauthorname)>
<!ELEMENT firstauthorname(#PCDATA)>
<!ELEMENT secondauthorname(#PCDATA)>
<!ELEMENT price(#PCDATA)>
<!ELEMENT publisher(#PCDATA)>
```

7.2 XML Schema

DTD 作为用户编写 XML 文档的结构标准是合适的，但作为计算机检查 XML 文档正确性标准却会勉为其难，其中主要问题如下。

(1) DTD 不具有名空间机制，难以检查所使用标记的合法性。

(2) 数据类型简单并且过少，不能适应较为复杂的情形。

(3) DTD 语法与 XML 文档本身语法有较大差异，不符合 XML 内在的半结构化特质，不方便计算机进行统一处理。

针对这种情况，W3C 推荐了一个 XML 模式(XML schema)标准作为检查 XML 文档正确性的依据。XML Schema 从 XML 文档中抽象出来，通常是先有 XML 文档，然后再有 XML Schema，这种半结构化数据特征使得它与传统数据模式有着较大区别。另外，XML Schema 采用与 XML 文档相同的形式定义文档结构，同时增加了对数据类型的支持。

【例 7-3】 例 7-1 的 XML Schema 如下。

```
<?xml version = "1.0" encoding = " ISO - 8859 - 1" standalone = "yes"?>
< schema xmlns.xsd = "http://www.w3c.org/2001/XMLSchema"
    xmlns.rgt = "http://www.sysu.edu.cn/CIT.XML/2016/book"
    elementFormDefault = "qualified"
    attuibuteFormDefault = "unqualified">
< element name = "book">
    < annotation >
        < documentation > book </documentation >
    </annotation >
    < complexType >
        < sequence >
```

```
            <element name = "bookname"type = "rgt:string"/>
            <element name = "author"minOccurs = "0"maxOccurs = "unbounded" >
                <complexType >
                        <sequence >
                            <element name = "firstauthorname"
                                    type = "rgt:string"/>
                            <elementname = "secondauthorname"
                                    type = "rgt:string"/>
                        </sequence >
                </complexType >
            <element name = "price"type = "rgt:short"/>
            <element name = "publish house"type = "rgt:string"/>
            </element >
      </sequence >
    </complexType >
</element >
</schema >
```

由此可以初步看出,XML Schema 具有如下基本特征。

(1) Schema 是一个 XML 文档。XML Schema 文档包含了标准的 XML 头<? xml version = "1.0"? >,这表示 Schema 本身就是一个 XML 文档。而任何 Schema 的根元素都必须是 Schema,它有一个或多个说明自己的属性。在这种情况下,Schema 的 namespace 定义属性(xmlns)会定义名称空间为 xs,它将用作文档中所有元素的根名称空间。实际上,人们是使用一套预先规定的 XML 元素和属性来创建 XML Schema,这些元素和属性定义了相应 XML 文档的结构和内容模式,由此得到的一套精巧规则指定了每个 Schema 元素或属性的合法用途,而这些规则却是使用 XML 来表示的。如果违反这些规则解析器会拒绝解析给定的 Schema 及任何相联系的文档。

(2) 具有健全的命名空间机制。一个 XML Schema 需要说明文档中所使用的名空间。一个 Schema 可以具有多个名空间,可在其中选择一个作为基本的默认名空间,凡未说明的都应属于这个名空间。例如,在 XML Schema 中,都需声明下述名空间(如声明,就是以其为默认名空间)。

```
xmlns:xsd = http://www.w3.org/2001/XML Schema
```

其中,xsd(XML schema definition)就是 W3C 公布的一个 XML Schema 的子名空间,用于规范 Schema 的定义。只要是属于这个名空间的标记和类型都可置以 xsd。

(3) 元素由名称和内容模型确定。Schema 通过对元素及元素相互之间关系的定义以实现对整个文档性质和内容的定义,其中,元素定义通过其名称和内容模型确定。这里,名称就是该元素名,内容模型实际上就是该元素所属的类型。就像在 C++中那样,可根据需要定义一个变量,但必须说明变量的类型。变量类型可有多种形式,可以是一个简单变量,如 C++内部指定的 bool、int、double 和 char 等简单类型,也可以是较复杂的类型,如 struct 或 class 类型。

(4) 具有丰富的元素类型。元素类型可以理解为元素的"取值"类型,这种取值可以是真正的"数据",也可以是具有结构的子元素。DTD 只提供 CDATA 、Enumerated、NMTOKEN、NMTOKENS 等 10 种内置(built-in)的基本数据类型。这些数据类型通常无法满足文档的

可理解性和数据交换的需要。XML Schema 则不同，它内置了 37 种基本数据类型，如 long、int、short、double 等常用的数据类型，并通过这些数据类型的派生机制（限制、列表与联合）定义新的数据类型以实现对用户自定义类型的支持。更为重要的是，XML Schema 还能描述具有结构的子元素类型。具体来说，XML Schema 将元素的类型（type）分为简单类型和复杂类型两种。简单类型被称为 simpleType。其特征是不包含子元素和属性；复杂类型被称为 complexType，其特征是不仅可以包含属性，而且可以在其中嵌套其他的元素，或者可以和其他元素中的属性相关联。

在 XML 中，定义元素需要声明元素的类型，下面分别讨论元素的简单类型和复杂类型。

7.2.1 简单类型

简单类型（simpleType）是一组由字符串表示的类型，其特征是不包含元素，它赋予 XML Schema 低级类型检查能力。

简单类型主要是在 XML Schema 内置（built-in）数据类型基础上使用 simpleType 来进行创建的。XML Schema 内置数据类型如表 7-1 所示。

表 7-1 内置数据类型

内置数据类型	说　　明
string	字符串数据
Boolean	二元类型的 True 或 False
date	历法日期，格式为 CCYY-MM-DD
dateTime	历法日期和时间
time	24 小时格式的时间，可根据时区调节
decimal	任意精度和位数的十进制数
integer	整数
float	标准的 32 位浮点数

基于内置数据类型进行派生的简单类型创建方式相当于对象关系数据类型中的类型生成器。XML Schema 支持的简单类型派生方式主要有限制类型（restriction）、列表类型（list）和联合类型（union）3 种。其中，限制是一种常用的类型派生技术；限制是通过对已经存在的简单类型的合法值加上某些限制条件而产生新的数据类型，这也可以看作是将原有类型值集合进行限制而得到的一个子集，以此方式来定义新的数据类型。简单类型派生机制如图 7-2 所示。

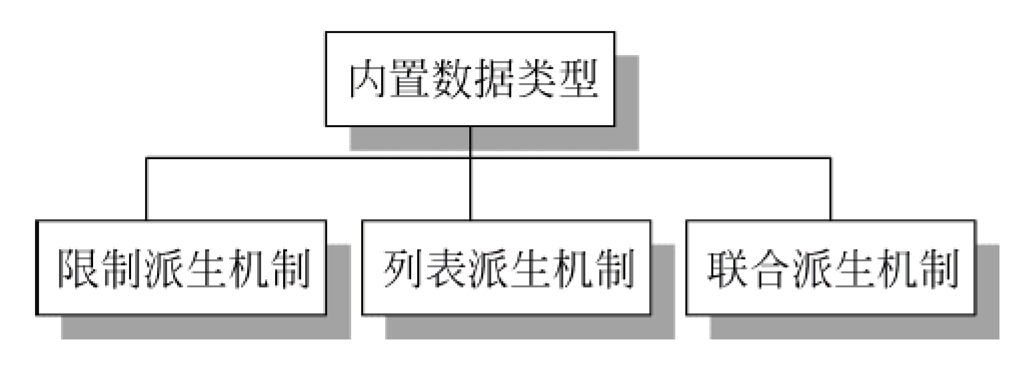


图 7-2 简单类型派生机制

【例 7-4】 restriction 方式的应用实例如下。

```
< simpleType name = 'Sku'>
```

```
        <restriction base = "rgt: integer">
        <minInclusivevalue = " - 10"/>
        <maxExclusivevalue = "10"/>
        </restriction>
</simpleType>
```

说明：限制派生通过 restriction 元素进行定义，由内嵌在 restriction 中的特定元素实行限制。被限定的数据类型称为基类型，通过 restriction 元素中< base >属性引用。在本例中，说明 Sku 类型实例应当是遵守下属限定条件"10＝Sku＜10"的 integer 类型的数据。

【例 7-5】 list 方式的应用实例如下。

```
<simpleType name =  "listOfinteger" >
    <list itemType = " integer" />
</simpleType>
```

说明：列表派生可以从一个原子类型得到一个该类型的列表数据类型。列表中所有数据项都应当是相同的数据类型。列表派生类型由 list 元素定义。本例表示派生出来的是一系列使用空格分离的 integer 类型数据的列表。

【例 7-6】 union 方式的应用实例如下。

```
<simpleType name = "DateOrDatetime">
    <union memberType = "rgt: date rgt: datetime"/>
</simpleType>
```

说明：联合派生类似于 C 语言中的联合(union)概念。联合允许在几类数据中选择所需要的一种类型，其中可以包括预定义类型和用户自定义数据类型。联合派生类型通过 union 元素定义。在本例中，表明当前类型实例允许出现 date 或 datetime 类型中的两者之一。

7.2.2 复杂类型

复杂类型(complexType)相对于简单类型具有更加强大的表述能力。复杂类型描述了一个确定元素的内容模式和属性列表，其中，内容模式是指元素内容的组成结构，而属性列表是指对元素某些特性的刻画。XML Schema 通过 complexType 元素定义复杂类型，既然是元素，就需要和某个元素类型相关联以说明某种结构类型，这种元素内容定义类型就是元素的内容模式。

复杂类型的内容模式可以分为空元素、简单内容、复杂内容和混合内容 4 种情形。

(1) 空元素：元素本身不包含任何子元素和文本数据。

(2) 简单内容：元素只包含文本数据，不包含子元素。

(3) 复杂内容：元素只包含嵌套元素，即包含子元素。

(4) 混合内容：元素既含有文本内容又含有嵌套元素。

注意，在上述 4 种情况中，元素都可以带有属性。

为了适应复杂类型中的各种情况，在 XML Schema 标准中，还需要下述一些内置的常用内容模式。

(1) sequence：在其定义范围之内的所有元素都必须按顺序出现，范围由 minOccurs 和

maxOccurs 指定。

(2) choice：其范围内有且只有一个元素必须出现。

(3) any：定义的任何元素都必须出现。

(4) simpleContent：这种复杂类型只包含了非嵌套元素。可以通过包含扩展元素的方式扩展先前定义的简单类型。

(5) complexContent：这种复杂类型只能包含其他元素。可以通过包含扩展元素的方式扩展先前定义的复杂类型。

(6) attribute：这种复杂类型只能包含命名属性。

上述这些内置的内容模式可以组成非常有用的组合器(Compositer)，应用在复杂类型的复杂内容模型和混合内容模型中。这些内容模式提供了一种功能强大的复杂数据类型定义机制，可以实现包括结构描述在内的复杂的数据类型。

1. 空元素

在不少应用情况下，需要声明一个元素不含有任何内容，此时元素通过其属性相对于其他元素的位置表达自身信息。

【例 7-7】 声明一个表达联系人信息的复杂类型 contactsType 的语句如下。

```
<comlexType name = "contactsType">
    <squence>
        <element name = "phone"minOccurs = "0">
            <complexType>
                <attribute name = "number"type = "rgt:string"/>
                </complexType>
            </element>
        </squence>
    </complexType>
```

说明：这里，phone 元素声明中包含复杂类型定义，而其中只有一个属性的声明，是一个空元素形式。

2. 简单内容

【例 7-8】 声明一个表达联系人姓名的复杂类型 fullName 的语句如下。

```
<element name = "fullName">
    <complexType>
        <simpleContent>
            <extension base = "rgt: string">
                <attribute name = "language"type = "rgt: language"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
```

说明：用以扩展的基本类型是内建的 string 数据类型。但简单类型并不只限于预定义类型，还可以引用用户通过 simpleType 元素自定义的新的简单类型。

3. 复杂内容

【例 7-9】 可以将例 7-8 改写为如下语句。

```
<element name = "phone"minOccurs = "0">
    <complexType >
        <complexContent >
            <restriction base = "rgt: anyType">
                <attribute name = "number"type = "rgt: string"/>
            </restriction >
        </complexContent >
    </complexType >
</element >
```

说明：在例 7-8 中，使用了 simpleContent 元素表明元素仅能包含简单内容而没有嵌套元素。在本例中，相应可以使用 complexContent 表明一个复杂类型只能包含嵌套元素而不能含有文本内容。当使用 complexType 元素声明 phone 元素时，如果没有对该元素不包含嵌套元素进行声明，则模式处理器将自动认为它仅包含简单内容。使用 complexContent 元素最常见方法是从一个现有类型中派生一个复杂类型。anyType 是所有内建模式类型的根类型，表示一个不受限制的字符和标记。由于 complexType 表明了 phone 元素仅包含元素内容，这个限制的作用是阻止该元素包含任何字符数据和标记。

4. 混合内容

【例 7-10】 定义 title 的 Schema 语句如下。

```
<element name = "a">
    <complexType >
        <simpleContent >
            <extension base = "rgt: anyType">
                <attribute name = "href"type = "rgt: anyURL"/>
            </extension >
        </simpleContent >
    </complexType >
</element >
<complexType name = "maxedText" mixed = "true">
    <choice minOccurs = "0" maxOccurs = "unbounded">
        <element name = "em"type = "rgt:token"/>
        <element ref = "a"/>
    </choice >
    <attribute name = "lang"type = "rgt: language"/>
</complexType >
<element name = "title" type = "rgt:marketedText"/>
```

说明：混合(mixed)内容模式的实例元素可以在各个子元素之间插入字符数据。由 mixed 特性，本例中相对应的实例文档可以表述为如下语句。

```
<title lang = "en">
      Being a
      <a href = "http://dmoz.org/Shopping/Pets/Dogs/">
          Dog
      </a >
          is a
      <em >
          Full - Time
```

```
        </em>
        Job
    </title>
```


7.2.3 元素与属性声明

1. 元素声明

XML Schema 中的元素声明实际上就是将涉及的元素名称与一个相关的简单结构类型与复杂类型联系起来。元素声明通常有“全局元素声明”和“局部元素声明”两种方式。

(1) 全局元素声明

全局元素声明就是将元素名称与相应元素类型(如内置类型)联系起来,其特征是作为相应元素的 type 属性元素出现。经过全局声明的元素可以在其他元素声明中进行引用。

例如,下述的元素声明将元素名称 Temporal XML 与类型 comment 连接在一起,因此属于全局元素声明。对于该实例文档中属于名空间 http://www.sysu.edu.cn/cit.2008/artical 中的所有 Temporal XML 元素都会起作用,即它们数据类型都是 comment。

```
<schema xmlns:xsd = http://www.w3.org/2001/XMLSchema
        xmlns; rgt = "http://www.sysu.edu.cn/cit.2008/artical">
    <element name = "Temporal XML" type = "rgt:comment"/>
</schema>
```

由本例可知,全局元素声明在表现形式上是元素类型声明紧接在元素名称声明之后,同在元素的开始标签内。

(2) 局部元素声明

局部元素声明主要作为内容模式定义的一个部分出现,如出现在 complexType 定义的内部,此时,元素类型声明使用子元素形式表示。局部声明元素只能存在于它们自己所处的局部范围中。

在下面代码中,元素开始标签中只有元素名称,而元素类型声明在开始标签之后,这种情况称为一个匿名类型定义(anonymous type definition)。由此定义的 comment 类型只能在元素 Temporal XML 范围内使用。

```
<schema xmlns:xsd = http://www.w3.org/2001/XMLSchema
        xmlns rgt = "http://www.sysu.edu.cn/cit.2008/artical">
    <element name = "Temporal XML">
        <simpleType>
            <restriction base = "rgt: string">
                <maxlength value = "1024" />
            </restriction>
        </simpleType>
    </element>
</schema>
```

2. 属性声明

属性声明实质上是将涉及的属性名称和一个简单类型在上下文环境中联系在一起。属性使用 attribute 元素进行声明。如同元素情形,属性有全局元素声明和局部元素声明之分。

例如，如果需要为元素 Temporal XML 增加一个 language 属性，其本身作为一个基于内置类型 string 的复杂类型。为此可通过 complexType、simpleContent 和 extension 3 个类型元素完成 language 属性的全局声明。

```
<element name = "remporal XML">
    <complexType>
        <simpleContent>
            <extension base = "rgt: string">
                <attribute name = "language"type = "rgt: language"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
```

7.3 XML 数据模型

作为一种特殊的文档文件，可以通过两个不同的视角对 XML 进行考查。

(1) 将 XML 看作纯文本：即字符流。例如，在关系数据库中存储 XML 时，可以通过前述的大数据类型将其作为关系表中某个属性的字符型属性值进行处理。这种方法没有充分利用 XML 中语义标签的基本功能。

(2) 将 XML 看作数据：也就是将 XML 中的标签作为语义识别依据，并将标签之间的嵌套关系作为相应的逻辑上的数据结构，这实际上就是从数据库数据模型观点审视处理，即将 XML 文档作为 XML 数据。但 XML 数据与一般数据库中的数据还是有很大差异，主要在于 XML 中的"实义数据"和"元数据"同一，以此为基础，常常是先有 XML 文档，再从中抽取出相应的数据模型。这种类型的数据就是半结构化数据。

7.3.1 半结构化数据

由于 Internet 及相关技术的快速发展，使得对网上信息的应用需求迅速增加。通常可从网络上得到海量网络数据，如何从这些海量数据中对所需要的信息部分进行准确定位和快速获取已成为网络数据管理中的迫切需求。现有网络数据查询技术多依赖于搜索引擎，基本上是通过关键字匹配进行查询，查询结果是一个由系统按照相关程度组成的网页信息集合，用户需要再进行逐一浏览和遍历查找，难以快速、准确和有效地获取查询结果。这种情况出现的根本原因在于网络数据大多是一种半结构化的数据(semistructured data)。

1. 概念和特征

传统数据模型(概念模型和逻辑模型)通常具有两个核心概念：结构模式和数据。例如，E-R 模型中的联系和实体集、关系模型中的关系模式和关系实例及面向对象模型中的类和对象实例集等。在传统模型中，结构和数据概念分别讨论，其意义在于较高层面上的结构模式和较低层面上的实际数据相互分离，使之具有某种意义下的相对独立性。大家已经知道，这种适当意义下的独立性对于简化系统结构和提升系统效率具有重要的作用。具有此种特征的数据模型通常称为结构化数据模型。如果将结构模式和数据这两个核心概念融合在一起，不再进行严格区分，则就需要考虑所谓半结构化数据模型。

相对于结构化数据，半结构化数据具有下述特点。

（1）自描述性。半结构化数据自身就可描述其具有的结构模式，而且通常是先有数据再考虑抽取相应的结构模式。

（2）不精确性。半结构化数据结构模式可以是不完全和不精确的，还会随时间及数据库的变动而变化。

（3）不规则性。半结构化数据模式没有规则的结构，不易进行统一规范的描述。

（4）非强制性。半结构化数据结构模式具有较大的灵活性，并不需要对所有数据都强制执行相应的结构模式，也就是说，数据可以没有相应的结构模式。

（5）模式复杂性。融合在数据中的结构模式可以非常复杂，相应结构模式描述规模可以大大超过数据本身的描述。

2. 半结构化数据模型

半结构化数据特征在于结构和数据的相互融合，数学中图形结构可以用于描述这种特性。Stanford University 的 TSMMIS 项目组提出一种基于图模型的半结构化数据模型（Object-Exchange Model，OEM），并将其应用于 lore 半结构化数据库管理系统。

1）OEM 基本概念

OEM 基本处理单元是对象，每个对象可以看作如下四元组。

（对象标签，对象类型，对象取值，对象标识）

对象的上述 4 个要素也称为对象的域。

（1）对象标签：描述对象名称的字符串。

（2）对象类型：描述对象取值的数据类型，根据所取的数据类型不同可将对象分为原子对象和复合对象两种类型。

（3）对象取值：对于原子对象而言，可以是基本数据类型及多媒体数据文件等不可再划分的原子数据类型；对于复合对象则是二元组（标记，子对象标识）的一个集合，其中标记描述对象与其子对象之间的关联。

（4）对象标识：每个对象所具有的唯一标识。

2）具根有向图

基于 OEM 的半结构化数据模型可看作一个简单、自描述和嵌套的对象模型，其数据可由一个具根有向图表示，即看作一个由某些弧段进行连接的结点（node）的集合，其中包括如下要素。

（1）叶结点。叶结点与适当的数据值相关联，表示结点和某个数据取值之间的关联，其中数据取值的数据类型为原子类型（如数值型或字符串型等）。

（2）内部结点。内部结点通过其发出的弧段与其他结点关联，内部结点发出的每条弧段都有一个标签（lable），用于指明弧段开始处结点与终止处结点的相互联系。

（3）根结点。根结点作为弧段出发点并表示整个数据文件，同时与集合中每个结点都有路径相连，根结点唯一。

有向弧段<u，v>可看作两个结点 u，v 之间的关联。在上述有向图中可以进行如下语义诠释。

（1）如果结点 u 表示一个对象，结点 v 表示 u 的属性，则弧段<u，v>表示对象 u 具有属

性 v,相应弧段标签可表示相应属性或属性域名称。

(2) 如果结点 u 和 v 都是对象,则弧段 $<u,v>$ 表示由 u 到 v 所具有的一种联系,弧段标签可表示给联系的名称。

(3) 如果结点 u 表示一个对象(属性),结点 v 表示一个具体数据,$<u,v>$ 表示对象(属性) u 取值为 v。

由 OEM 可知,半结构化数据是自描述的(self-describing),结构模式与数据结合一体,弧段标签说明末端结点在始端结点中的角色,这实际上就是将模式结构与实际数据通过同一表现形式进行整合描述。在结构化数据中,结构模式固定并与数据分离,数据角色信息隐含在结构模式中。关系数据等结构化数据优势在于其较高的操作效率,半结构化数据优势在于其所具有的灵活性。由于灵活性,半结构化数据适合描述多个不同结构模式但具有模型相似性的数据,即能够实现数据集成;由于模式与数据的结合性,半结构化数据还适合于文档性数据的描述,如 Web 网络上的共享信息。

7.3.2 数据关系与数据结构

从 XML 本身来源考虑,一个规范化的 XML 文档就是一个基本的 XML 数据,因此,XML 文档就是 XML 数据的基本形式;从 XML 自身具有的嵌套结构考虑,又可以将 XML 文件看作一棵具有适当约束的树形结构,因此 XML 树就构成了 XM 的基本数据模式。实际上,作为一种半结构化数据的表现形式,XML 数据模型可以看作是 OEM 数据模型的具体实现。

1. XML 数据结构关系

由 XML 数据本身组成考虑,其包含的信息可分为内容信息和结构信息两个方面。前者主要体现在元素和属性的文本取值,后者主要体现在元素之间的嵌套关系、引用关系和结点顺序关系。

1) 嵌套关系

按照数据管理观点,反映 XML 数据模式信息之间的结构关联最为重要,只要理解和掌握了元素之间的结构关系,才可能进行 XML 数据的有效查询。这种基于嵌套关系的结构关联通常表现为下述两种情况。

(1) 包含关系:如果标记单元 u 所管理的所有数据片段都是标记单元 v 所管理数据片段的组成部分,则称标记单元 v 包含单元 u。在 XML 树形结构中就表现为 v 和 u 之间的祖先/子孙关联。

(2) 邻近关系:表明同一标记单元所管理的文本区域内字符串之间的距离,此时,在 XML 树形结构中表现为文本结点之间的兄弟关系。

2) 引用关系

从应用角度考虑,数据对象之间的相互引用具有重要的意义。通过引用,可以大大减少数据的重复存储,同时也能更加清晰地表达数据对象之间的相互关系。在 XML 数据中,通常由 IDREF 属性或 XLink 语法实现元素之间的相互引用。

3) 顺序关系

给定了一个 XML 数据,实际上就隐含了其中标记结点之间的一种顺序,通常将其称为 XML 顺序。XML 顺序概念在数据查询(数据索引)中发挥着重要作用,这是因为标记结点

相互之间顺序结构是设计各种查询算法的基础,同时查询结果通常是被查询 XML 数据的一个片段,其中标记结点顺序应当与原有顺序一致。

一般来讲,XML 可以看作是一种半结构化数据的描述语言。事实上,如果对于文档中的标签二元组< E1 ></E1 >创建一个结点 n_1,而对其中紧接嵌套的下层标签二元组< E2 ></E2 >创建结点 n_2 后,引入一条连接 n_1 和 n_2 的弧段,由此就得到一个从 XML 文档集合到半结构化数据集合的映射,即一个 XML 文档对应着一个半结构化数据。为了将这种思想规范化和标准化,2003 年,W3C(World Wide Web Consortium)为 XML 数据模型发布了工作草案 XQuery1.0 和 XPath2.0 Data Modle,其中提出了基于 XQuery 的 XML 查询数据模型(XML query data model)概念,由于这种模型主要着眼于数据的查询处理,可看作是 XML 数据模型的处理形式,它实际上是一种半结构化数据的有向图形结构。

2. XML 数据有向图

XML 有向图模型 G 定义为五元组:G=(VG,EG,∑G,ch1,lab,val,Vr)

(1) Vr:G 中根结点,对应于 XML 文档中的根结点。

(2) VG:G 中除了根结点外所有数据结点集合,其中结点 v 有元素(属性-值)结点和文本结点(元素的文本内容)两种类型。

这两类结点都具有结点编码(v-nodecodeld)和结点标签(v-lable)两个基本属性。

(3) EG:为 G 中所有边的集合,其中包含两种不同类型的边。

① 引用边:表示元素间引用关系的引用边(reference edge),通过 ID/IDREF 属性或 XLink 语法[[DMO01]实现。

② 树边:表示元素间父子关系和元素-属性关系的树边(tree edge)。

(4) ∑G:G 中所有标签集合。在 G 中,如果在结点的连线上出现边标记,则该标记就是其下端结点(属性结点或文本结点)的元素名称。分别以 E 表示元素名称集合,以 S 可解析的字符串即#PCDATA 的指代集合,以 A 表示属性名称集合。

(5) G 中 chl、lab 和 val 分别表示下述 3 个函数。

① chl 函数:子结点函数,用于计算子结点,如给定一个结点 n,chl(n)表示 n 的所有子结点集合。

② lab 函数:由 VG 到 ΣG 上的映射函数,即对于每个类型 n,lab(n)表示为 n 赋予的一个标签;元素类型标签为该元素名称,属性结点为该属性名称,#PCDATA 类型标签为 S 中的元素。

③ val 函数:类型值函数,即对于元素类型,val 返回其 ID 值;对于属性结点,val 返回结点的属性值;对于文本结点,val 返回结点的字符串值。

一个电影 XML 数据的有向图数据模型实例如图 7-3 所示,其中实线代表树边,虚线代表引用边,结点按照深度优先遍历编号。

3. XML 数据有向树

还可以将 XML 文档表示为一棵带有标签有向树:T=(VT,ET,TRoot, ∑)

(1) VT:表示 T 中所有结点的集合。

(2) ET:表示 T 中所有边的集合。

(3) TRoot:为 T 的根结点。

(4) ∑:所有结点所带标签的集合。

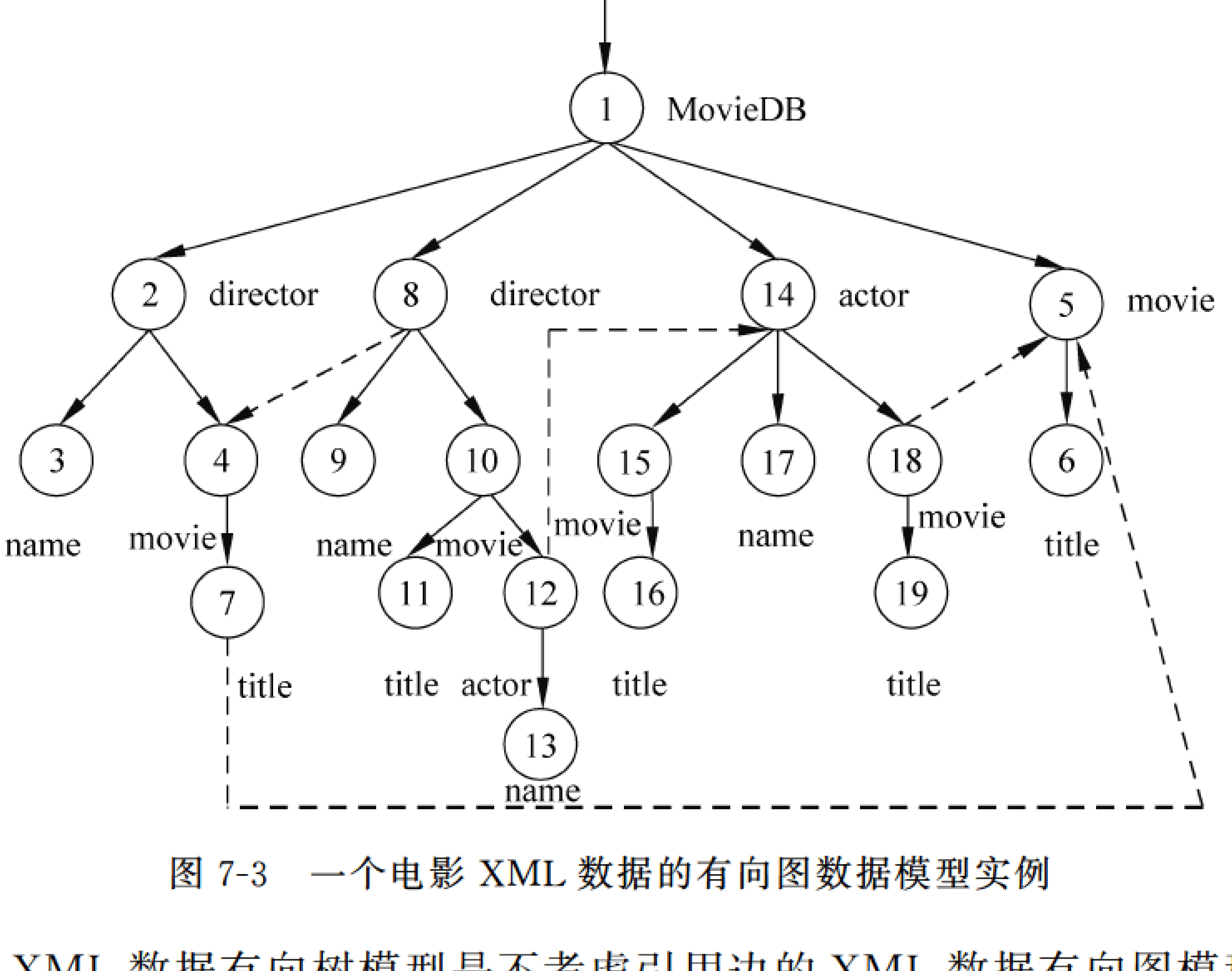


图 7-3　一个电影 XML 数据的有向图数据模型实例

由此可知，XML 数据有向树模型是不考虑引用边的 XML 数据有向图模型，即 XML 数据有向图模型的简化形式。

注意，XML 数据中可能存在实体，因而会在同一个文档中出现对相应实体的引用，通常需要将 XML 数据建模为有向图。但从理论上来讲，通过将被引用结点进行“重复”设置就可以将有向图转换为相应的有向树形结构。为了表述简便和突出主线，本章以下主要考虑的是 XML 的有向树模式，其实，作为图的特殊形式，研究树的情形也是研究图情形的重要基础。

一个出版物 XML 的树模型的实例如图 7-4 所示。

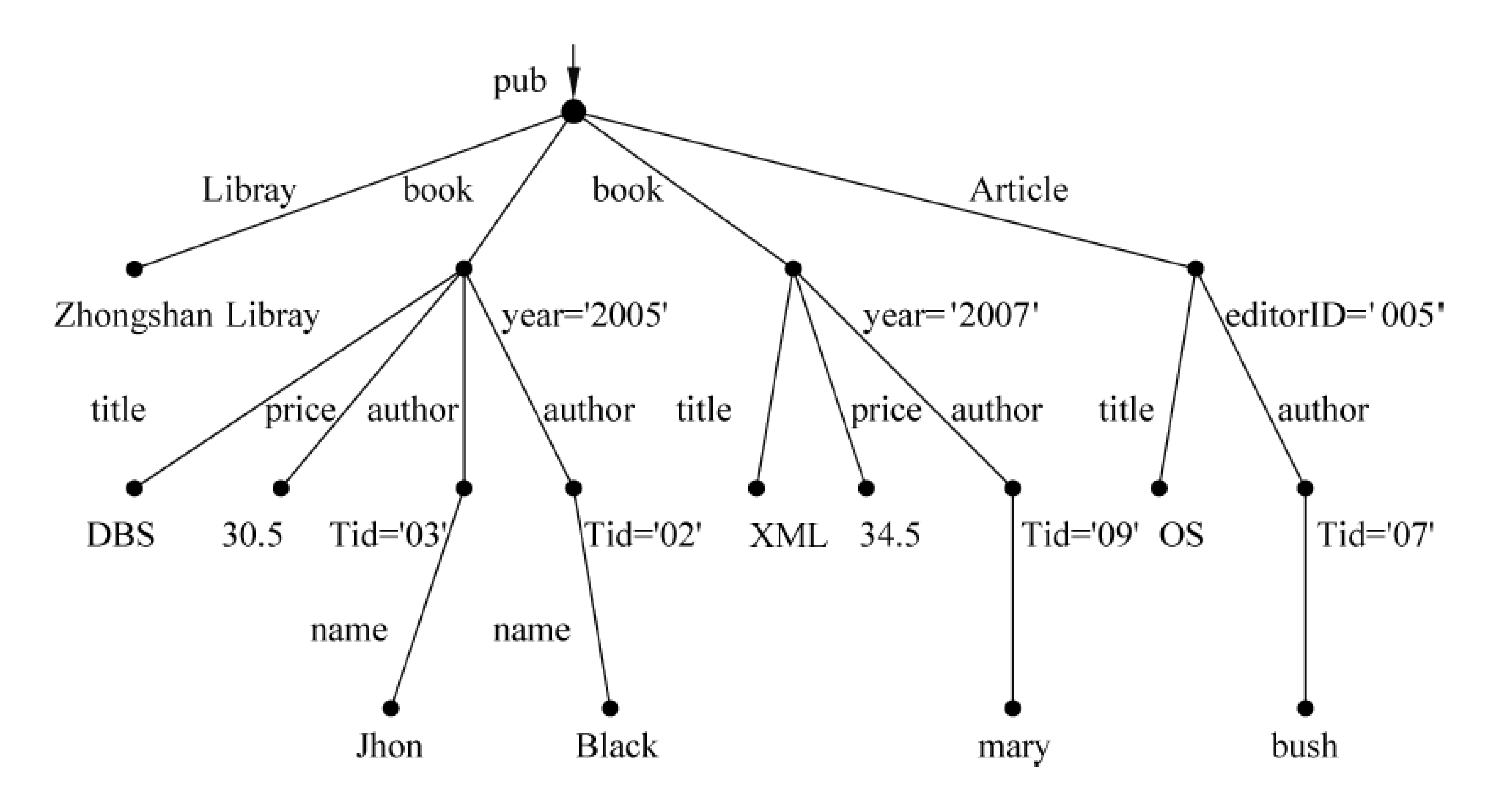


图 7-4　XML 有向树数据模型实例

4．结点路径和标签路径

在具有嵌套结构的层次化数据模式中，数据元素的路径是实施相应数据查询与处理的基本概念。对于基于树形结构的 XML 数据查询而言，也需要引入结点路径和标签路径概

念。设 T 为给定的一个 XML 有向树数据模式。

(1) 结点路径：一个形如 $v_0v_1\cdots v_m$ 的数据结点序列称为一条结点路径(Node Path)并记为 P_m，其中$(v_{i-1},v_i)\in ET(0\leqslant i\leqslant m)$。从 T 的根结点 root 开始到叶结点结束的结点路径被称为 T 的完全结点路径，简称 T 的完全路径(Complete Path)。

(2) 标签路径：一串以“/”相隔的连续标签组成标签序列 $l_0/1_1/\cdots/l_k$ 的表达式称为一条标签路径(Label Path)，且记作 P_r，其中 $l_i\in\sum(0\leqslant i\leqslant k)$。从 T 中某一结点 v 开始向根结点方向伸展的长度为 k 的标签路径被称为 v 在 T 中的 k 阶输入标签路径。

对于任意两条结点(标签)路径 P_1 和 P_2，若 P_1 是 P_2 的一部分，则称 P_1 是 P_2 的子路径，记作 $P_1\subset P_2$。

对于任意标签路径 P，若结点 v 在 T 中的某一 k 阶输入标签路径等于 P，则称 v 是 P 的目标数据结点(Target Data Node)。P 的所有目标数据结点组成的集合被称为 P 的目标数据结点集，即 P 的查询结果。若 P 的目标数据结点集不为空，则称 P 存在于 T 中。

对于 k 阶标签路径 $l_0/1_1/\cdots/l_k$ 和结点路径 $v_0v_1\cdots v_m$，如果$(k=m)\wedge(\text{label}(v_i)=l_i)$，则称结点路径 P 和标签路径 L 相互匹配，其中，$\text{label}(v_i)$表示结点 v_i 的标签。

需要注意，通常情况下一个标签路径可以对应多条结点路径，这也为通过标签路径索引结点路径提供了可能。

利用分隔符“/”、二分符“|”、可选符“?”和循环符“*”可定义基本正则路径表达式 R (basic regular path expression)。若 R 中只允许出现标签和分隔符“/”，则 R 被称为简单路径表达式(simple path expression)，简称简单路径。简单路径等价于标签路径。

7.4 XML 数据查询

数据查询是任何数据管理系统的关键性操作。由于 XML 文档可以建模为有向树或具根有向图的数据模式，可以依据对于树或图的遍历方式实现“遍历查询”。同时由于一般树和图的性质和 XML 数据树(图)的自身特性，还能够实现“非遍历查询”，这就是 XML 数据索引。需要注意的是，XML 数据索引实际上还是一种基于查询的数据存储结构。

7.4.1 遍历查询

作为一种树形或图形结构，实现 XML 数据查询的技术基础是路径表达式。

1. 路径查询表达式

XML 数据基本形式是 XML 文档；XML 数据模式是按照元素之间的相互关系建立的树形结构；XML 查询的基础是基于 XPath 的查询表达式，而这种查询表达式的基本构件就是路径表达式。XML 查询过程中使用的路径表达式通常可分为下述几种情况。

(1) 根据复杂程度，分为线性路径表达式和分枝路径表达式。

(2) 根据路径匹配的始点，分为绝对路径表达式和相对路径表达式。

(3) 根据通配符出现与否，分为简单路径表达式和复杂路径表达式。

基于 XPath 路径表达式主要描述如下 3 种类型的约束条件。

① 结构约束条件：主要是表达式中出现的元素之间的包含关系和文档位置关系。例如，一个基于 XPath 表达式 a//b[c/text()='databases']就表示了结点 a 和 b 之间就存在着

祖先/子孙的包含关系。

② 标签约束条件：结点的标签名称限制。在上述查询表达式中，出现的 3 个元素标签名称分别为 a、b 和 c。

③ 文本约束条件：元素本身及属性的文本取值。在前述表达式中，元素 c 的取值限制为'databases'。

一种有效的 XML 数据查询技术应当对相应基于 XPath 查询表达式中的上述 3 种约束进行快速甄别定位。上述 3 种约束中，“标签约束”和“文本约束”都可以看作是“语义”约束，通过使用常规的谓词选择或语义索引完成相应查询；结构约束来源于 XML 中元素自身的嵌套结构特征，不易使用常规方式进行处理，由此成为 XML 查询研究的主体技术。

2. DOM 和 SAX

XML 查询技术本质上就是对 XPath 查询表达式的有效响应。文档对象模型(Document Object Model，DOM)和 XML 简易 API(Simple API for XML，SAX)是 XML 数据处理过程中的两种接口规范。借助于 DOM 和 SAX 可以实现基于 XPath 表达式的 XML 数据查询。下面通过一个简单路径处理表明 DOM 和 SAX 基于深度优先的 XPath 查询过程。

输入：XML 文档，简单路径表达式查询语句：Q=/a/b/c/d。

输出：XML 中满足 Q 的结点。

基本操作如下。

(1) 先进行深度遍历 XML 顺序得到的结点集合，再对该集合中结点判断其标签路径是否符合查询模式。

(2) 如果得到布尔值为 True，则输出该结点，同时跳过该结点所辖区域。

由于复杂查询可以分解为简单查询表达式，因此上述简单表达式的操作方式也是复杂查询的基本操作方式。

7.4.2 查询语言 XPath

XPath 是 W3C 定义的标准并予以正式推荐的 XML 查询语言，设计主旨是为了在 XML1.0 或 XML1.1 文档结点树中定位结点。XPath 最初作为对 XSLT 和 XPointer 语言的补充，但现已成为广为流行的独立语言，其优势在于一个 XPath 表达式可用于替代多行 DOM API 代码。

1. 结点与路径表达式

XPath 具有 7 种结点类型：元素、属性、文本、名空间、处理指令、注释及文档(根)结点。在 XPath 中，一个 XML 文档是被作为有结点组成的树形数据结构，而文档结点作为相应的根结点。

【例 7-11】 设有记为文件名 bookstore2.xml 的如下 XML 数据。

```
<?xml version = "1.0" encoding = "ISO - 8859 - 1"?>
< bookstore >
    < book >
      < title lang = "eng"> Introduction of XPath </title >
      < author > A. Raul </author >
      < year > 2016 </year >
```

```
07.        <price>$30.50</price>
08.      </book>
09.      <book>
10.        <title lang="eng">Principal on XQuery</title>
11.        <author>J. White</author>
12.        <year>2017</year>
13.        <price>$26.00</price>
14.      </book>
15.  </bookstore>
```

XPath 使用路径表达式来选取 XML 文档中的结点或结点集。结点是通过沿着路径(Path)或步(Steps)来选取的。

(1) 绝对路径：如果“/”处在 XPath 表达式开头则表示文档根元素(表达式中间作为分隔符用以分割每一个步进表达式)，如/messages/message/subject 是一种绝对路径表示法，它表明是从文档根开始查找结点。

(2) 相对路径：假设当前结点是 book 结点，则路径表达式 book/price 中“book”前没有出现“/”，这种表示法表明从当前结点开始查找并称为相对路径。

XPath 中查找基于遍历方法如下。

在绝对路径情形：只有一种遍历途径，即由根结点沿路径“向下”行进直到目标结点。

在相对路径情形：由“当前结点”开始，有“上下左右”多个方向可供选择，此时就需要考察“表达式上下文”。

表达式中的上下文(context)：表示一种环境，以明确当前 XPath 路径表达式处在什么样的环境下执行。如前所述，在 XPath 中，同一路径表达式是处在根结点操作环境和处在对某一特定结点操作的环境下执行所获得的结果可能完全不同，此时基于路径表达式的计算就取决于它所处的上下文。实际上，“上下文”可以看作是其他结点关于“当前结点”的结构关系描述，这种关系主要有下述几种情况。

(1) 当前结点(./)：如./book 表示选择当前结点下的 book 结点集合，也等同于通常的“特定元素”，如 book。

(2) 父结点(../)：如../book 表示选择当前结点父结点下的 book 结点集合。

(3) 根元素(/)：如/bookstore 表示选择文档的根结点 bookstore 结点集合。

(4) 根结点(/*)：这里*代表所有根结点，但根结点只有一个，所以只表示根结点。/*的返回结果和/bookstore 返回的结果一样都是由一个 bookstore 结点组成的单元素集合。

(5) 递归下降(//)：表示由匹配选择当前结点选择文档中的结点，而不考虑它们的位置。例如，路径表达式//book 选取所有 book 子元素，而不管它们在文档中的位置。又如，路径表达式 bookstore//book 选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。

(6) 选取属性@：如路径表达式//@lang 选取名为 lang 的所有属性。

2. 路径表达式基本类型

为了提高查询效率，XPath 引入了各类路径表达式。

1) 基于谓语路径表达式

谓语(predicates)用来查找某个特定的结点或包含某个指定的值的结点。

在 XPath 中，路径表达式中的谓语被嵌在方括号[.]中。XPath 的谓语即筛选表达式，类似于 SQL 的 WHERE 子句。

下面列出了带有谓语的一些路径表达式及表达式的结果。

(1) 路径表达式：/bookstore/book[1]

查询结果：选取属于 bookstore 子元素的第一个 book 元素。

(2) 路径表达式：/bookstore/book[last()]

查询结果：选取属于 bookstore 子元素的最后一个 book 元素。

(3) 路径表达式：/bookstore/book[last()-1]

查询结果：选取属于 bookstore 子元素的倒数第二个 book 元素。

(4) 路径表达式：/bookstore/book[position()< 3]

选取最前面的两个属于 bookstore 元素的子元素的 book 元素。

(5) 路径表达式：//title[@lang]

查询结果：选取所有拥有名为 lang 的属性的 title 元素。

(6) 路径表达式：//title[@lang='eng']

查询结果：选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。

(7) 路径表达式：/bookstore/book[price>35.00]

查询结果：选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。

(8) 路径表达式：/bookstore/book[price>35.00]/title

查询结果：选取 bookstore 元素中的 book 元素的所有 title 元素，其中 price 元素值须大于 35.00。

2) 基于通配符路径表达式

如果路径表达式中具有未知结点，XPath 则可以采用基于通配符的路径表达式。

XPath 通配符可用来选取未知的 XML 元素。

(1) *：匹配任何元素结点。

(2) @*：匹配任何属性结点。

(3) node()：匹配任何类型的结点。

下面列出一些带通配符的路径表达式及这些表达式的结果。

(1) 带通配符表达式：/bookstore/*

查询结果：选取 bookstore 元素的所有子元素。

(2) 带通配符表达式：//*

查询结果：选取文档中的所有元素。

(3) 带通配符表达式：//title[@*]

查询结果：选取所有带有属性的 title 元素。

3) 基于并列选择路径表达式

如果需要同时选取若干并列路径，则 XPath 可以采用并列的多路径表达式。

通过在路径表达式中使用“|”运算符以选取若干个路径。

下面列出了一些并列路径表达式及这些表达式的结果。

(1) 并列路径表达式：//book/title | //book/price

查询结果：选取 book 元素的所有 title 和 price 元素。

(2) 并列路径表达式://title | //price

查询结果:选取文档中的所有 title 和 price 元素。

(3) 并列路径表达式:/bookstore/book/title | //price

查询结果:选取属于 bookstore 元素的 book 元素的所有 title 元素,以及文档中所有的 price 元素。

7.4.3 查询语言 XQuery

XQuery 是一个用于 XML 查询的语言,某些情况下也被称为 XML Query。W3C 在 2007 年 1 月 23 日颁布 XQuery 1.0 为推荐标准。

XQuery 被设计用来查询以不同形式出现的 XML,包括 XML 文档形式和 XML 数据库文件形式。从 XML 数据库角度来说,XQuery 对于 XML 的意义就类似于 SQL 对于关系数据表的意义。作为一种 W3C 标准,XQuery 基本功能就是从 XML 文档数据中查找和提取元素及属性,现在已经得到几乎所有数据库如 DB2、Oracle 和 SQL Sever 等的支持。

1. FLWOR 表达式

XQuery 具有 SQL 的查询风格,一方面反映了人们使用数据库管理 XML 文档的认知和努力,另一方面也表现了 SQL 的巨大影响。其中最具特色的就是 XQuery 中的 FLWOR 查询语句。

1) 基本查询子句

如同人们将 SQL 基本查询语句看作是 SELECT 表达式,XQuery 基本查询语句也可以说就是 FLWOR 表达式。这里,FLWOR 分别是 for、let、where、order by 和 return 5 个查询子语句的首字母,表示了五类基本的查询子句。

(1) 遍历语句 for…in…。例如,for $x in doc("books.xml")//book,在所有的 book 元素中遍历,其中 $x 是临时变量。

(2) 赋值语句 let。例如,let $x:=3,把 3 赋给 x;let $x:=(1 to 5),把 x 赋值为 1~5 (1 2 3 4 5)。

(3) 条件判断语句 where。例如,where $x>30 and $x<60。可用 and 或 or 把多个条件连接。

(4) 排序语句 order by。例如,order by $x,按变量 x 的值排序,可以使用“,”进行多重排序。

(5) 返回语句 return。返回查询结果,例如,return $x。

【例 7-12】 设有记为文件名 bookstore3.xml 的如下 XML 数据。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2015</year>
        <price>$30.00</price>
    </book>
    <book category="CHILDREN">
```

```
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2016</year>
        <price>$29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">XQuery Kick Start</title>
        <author>James McGovern</author>
        <author>Per Bothner</author>
        <author>Kurt Cagle</author>
        <author>James Linn</author>
        <author>Vaidyanathan Nagarajan</author>
        <year>2016</year>
        <price>$49.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2018</year>
        <price>$39.95</price>
    </book>
</bookstore>
```

使用 FLWOR 从 "books.xml" 选取结点的查询表达式如下。

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

上述表达式表示需选取 bookstore 元素下的 book 元素下所有的 title 元素,并且其中的 price 元素的值必须大于 30.00。

(1) for 语句:把 bookstore 元素下的所有 book 元素提取到名为 $x 的变量中。

(2) where 语句:选取了 price 元素值大于 30.00 的 book 元素。

(3) order by 语句:定义了排序次序。将根据 title 元素进行排序。

(4) return 语句:规定返回什么内容。在此返回的是 title 元素。

查询结果如下。

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

如果还需要对查询结果进行排序,则可使用下述 FLWOR 表达式。

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
```

查询结果为:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

需要指出，上面第一个 FLWOR 表达式和下述使用 doc()函数的 XPath 路径表达式选取结果相同：

```
doc("books.xml")/bookstore/book[price>30]/title
```

2) XQuery 语法规则

(1) 基础语法规则：XQuery 基础语法规则主要有下述情形。

① 字符及字符串规则：XQuery 表达式对于字符的大小写敏感，同时 XQuery 字符串值可使用单引号或双引号。

② 元素与变量规则：XQuery 元素、属性及变量必须是合法的 XML 名称，XQuery 变量由“$”并跟随一个名称来进行定义，如 $bookstore。

③ 注释规则：XQuery 注释被(：和：)分割，如(：XQuery 注释：)。

(2) 条件表达式 XQuery 条件表达式中可以使用“if-then-else”。

【例 7-13】 设有如下查询表达式。

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
      then <child>{data($x/title)}</child>
      else <adult>{data($x/title)}</adult>
```

需要注意得是，if 表达式后的圆括号和 else 都是必需的，不过也可仅写 else()。

上述表达式查询结果为：

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>Learning XML</adult>
<adult>XQuery Kick Start</adult>
```

(3) XQuery 比较：在 XQuery 中，有两种方法来比较值。

① 通用比较：=、!=、<、<=、>、>= 。

② 值的比较：eq、ne、lt、le、gt、ge 。

通用与值比较 XQuery 表达式如下。

通用比较：$bookstore//book/@q > 10 如果 q 属性的值大于 10，上面的表达式的返回值为 true。

值比较：$bookstore//book/@q gt 10 如果仅返回一个 q，且它的值大于 10，那么表达式返回 true。如果不止一个 q 被返回，则会发生错误。

3) 获取文档和结点

XQuery 使用函数来提取 XML 文档中的数据。

获取 xml 文档需使用 doc()函数，如 doc("books.xml")，其中，doc()用于打开"books.xml"文件。

得到 doc 之后 XQuery 就可使用 XPath 路径表达式在 XML 文档中通过元素进行导航而取值，如 doc("books.xml")//book/title，取得当前文档中所有 book 元素下的 title 元素。

另外，也可以使用限定词来取值，如 doc("books.xml")//book[price=30]/title，取得当前文档中所有价格等于 30 的书的 title 元素。

下面的路径表达式用于在 books. xml 文件中选取所有的 title 元素：

```
doc("books.xml")/bookstore/book/title(/bookstore 选取 bookstore 元素,/book 选取 bookstore
元素下的所有 book 元素,而 /title 选取每个 book 元素下的所有 title 元素)
```

此时 XQuery 可获得如下结果。

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

下面的谓语用于选取 bookstore 元素下的所有 book 元素，并且所选取的 book 元素下的 price 元素的值必须小于 30。

```
doc("books.xml")/bookstore/book[price<30]
```

上面的 XQuery 可提取到下面的数据。

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

4）引用输入文件中的元素和属性

XQuery 允许在结果中引用输入文件中的元素和属性。

设有如下表达式。

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

此时，会在结果中引用 title 元素和 lang 属性。

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

以上 XQuery 表达式返回 title 元素的方式和它们在输入文档中被描述方式相同。

2. XQuery 函数

XQuery 1.0、XPath 2.0 及 XSLT 2.0 共享相同的函数库。XQuery 含有超过 100 个内建的函数。这些函数可用于字符串值、数值、日期及时间比较、结点和 QName 操作、序列操作、逻辑值等。用户也可在 XQuery 中定义自己的函数。

1）XQuery 内建函数

XQuery 函数命名空间的 URI：http://www.w3.org/2005/02/xpath-functions。

函数命名空间的默认前缀为 fn:。

需要注意的是，函数经常被通过 fn: 前缀进行调用，如 fn: string()。但是，由于 fn: 是

命名空间的默认前缀，因此函数名称不必在被调用时使用前缀。

函数调用可与表达式一同使用。

(1) 在元素中调用。

```
<name>{uppercase($booktitle)}</name>
```

(2) 在路径表达式的谓语中调用。

```
doc("books.xml")/bookstore/book[substring(title,1,5) = 'Harry']
```

(3) 在 let 语句中调用。

```
let $name:=(substring($booktitle,1,4))
```

2) XQuery 用户定义函数

用户也可在查询中或独立的库中自定义函数，相应语法格式如下。

```
declare function 前缀:函数名($参数 AS 数据类型)
  AS 返回的数据类型
{
   (: ...函数代码... :)
};关于用户自定义函数的注释:
```

需要注意的是，应当使用 declare function 关键词，同时，函数名须使用前缀；参数的数据类型通常与在 XML DTD 或 Schema 中定义的数据类型一致；函数主体须被大括号包围。

一个在查询中声明的用户自定义函数如下。

```
declare function local:minPrice(
  $price as xs:decimal?,
  $discount as xs:decimal?)
  AS xs:decimal?
{
let $disc:=($price * $discount) div 100
return ($price - $disc)
};
(: Below is an example of how to call the function above :)
<minPrice>{local:minPrice($book/price, $book/discount)}</minPrice>
```

随着 XML 应用的普及，对 XML 文档查询的要求也就越来越高。如果不对 XML 文档建立索引结构，那么针对 XML 数据的任何查询都很可能导致对整个文档树的遍历。随着 XML 数据集的增大，这种遍历所花费的开销是不可忍受的，XML 索引结构的提出正是为了提高查询的效率，在速度与准确性两方面为查询提供更大的灵活性，通过减少访问那些与查询不相关的数据集来实现快速查询。对 XML 数据建立有效的索引，是关乎 XML 数据处理性能的重要因素。

7.4.4 遍历查询存在的问题

基于规范接口的 DOM 和 SAX 查询方式会出现下述两个问题。

(1) 路径多次搜索：进行查询语句中结构关系匹配过程中，需要遍历结果结点集合中

元素所涉及的全部路径,对查询效率影响较大。

(2) 结点重复访问：对于 XML 中具有相同标签路径的结点,需要进行多次重复访问。

SAX 和 DOM 都是基于文档处理的 XML 查询方式,依据的查询技术主要深度优先遍历,因此从本质上来说,还是属于"遍历查询"的范畴,存在着数据查询方面的技术缺陷。下面以图 7-5 所示的 XML 模式予以说明。

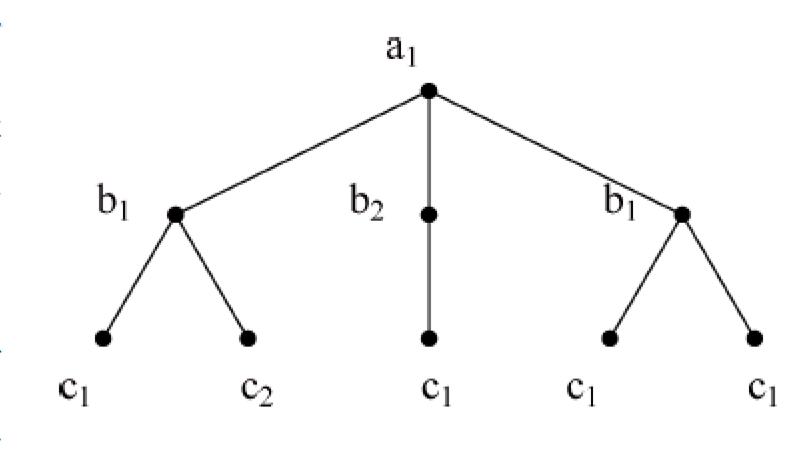


图 7-5　XML 树实例

(1) 访问无关结点：在 XML 数据中,如果采取遍历方式寻找满足查询相应结构关系的数据结点,则可能需要访问大量在查询表达式中不出现的结点。例如,对于查询表达式 $a_1//c_1$ 来说,按照遍历方式就需要访问 b_1 和 b_2 等在表达式中没有出现的结点,从某种意义上来看,遍历访问了"无用"的结点。

(2) 重复访问标记相同结点：XML 文档中,通常具有大量完全相同的标记路径,如在图 7-5 中的标记路径"$a_1/b_1/c_1$"就出现了两次。在遍历方式中,这样的路径不管有多少次,都需要逐一访问,即相同标记路径都要访问多次,这实际上是一种"冗余"访问,从而导致较低的查询效率。如果能将同样路径的结点汇集到同一结点中,就可大幅提高搜索效率。

(3) 产生大量随机 I/O ：如果在存储过程中,XML 数据树的叶结点不能放在同一数据块中,则相应查询就会涉及大量不同数据块访问,产生很多的随机 I/O。例如,对于查询表达式 $a_1//c_1$ 来说,如果图 7-5 中叶结点不在同一数据块中,则相应遍历查询就需访问多个数据块。

针对 SAX 和 DOM 不足,人们提出 XML 数据索引技术。

7.5　XML 数据索引

XML 文档通过数据建模后,具有如下基本特性。

特性 1：可以表示为有向树或具根有向图,而对于树或图都存在着各类有效的遍历方式。

特性 2：其中一条标签路径可以对应多条不同的结点路径。

(1) 由特性 1：通过适当遍历方式将树或图中所有结点排序,同时赋予某种带有结点结构与位置信息的适当编码,这样就将"结点"集合转换为"编码"集合。由于编码集合"有序",在利用经典索引技术(如 B^+ 树等)时就可以建立起基于"结点记录"的索引。

(2) 由特性 2：可以将具有相同"标签路径"的"结点路径"进行归并聚类,这里的关键问题是归并聚类标准即"等价关系"的设定,得到的等价类就是所谓的"结构摘要",由此就可以建立起基于"结构摘要"的索引。

本节简要介绍基于上述两种类型的 XML 数据索引方法。

7.5.1　基本考量与分类

建立任何一种索引技术都有其自身的基本考量,即索引操作的实现需要达到怎样的查询目标。同时,基于某种数据对象可能存在不止一种数据索引方法,对这些索引进行适当分类以昭显其内在特质和使用边界,这对于相关原理与技术的深入研究具有基本的价值。

1. 索引考量要素

一种 XML 索引技术能够在数据查询过程中有效地发挥作用，需满足一些基本技术要求。

1）基于查询要求

不论 XML 索引采用何种形式，其实际设计与实现都须考虑 XML 查询的基本特征——结构关系的保存和根据结构信息快速计算结点之间的位置关系。如前所述，相对于常规数据索引查询，XML 查询涉及较多内容，主要包括下述两个方面。

（1）获取结构关系。传统索引技术主要是根据数据的值来定位数据记录，本质上并不关注数据关系的特定表示模式即数据记录间的逻辑关系。XML 数据查询基本要求却是需要根据模式特征（路径表达式形式描述的结构关系）的输入提取符合该模式的数据，因此 XML 索引实际上就是一种适用于模式匹配的技术，而针对路径表达式的相关设计也就成为索引需要考虑的重点内容。XML 查询中模式匹配的主要问题是对 XML 数据中结构关系的表达，考虑如何利用已有的索引技术为高效获取符合结构关系（如包含关系）的数据集合提供支持。

（2）保持顺序关系。在具体 XML 文档中，各个元素之间都具有一定顺序限制，这种顺序就称为 XML 文档顺序。元素在文档中出现的顺序取决于对 XML 树的适当遍历方式是先序遍历、中序遍历还是后序遍历。例如，当查询关系存储的 XML 数据时，由于关系模式不存在顺序概念，因此需要在分解 XML 的过程中考虑如何确保查询的结果仍然符合结果元素集在原 XML 数据中的顺序关系，与这些元素在原 XML 文档先序遍历顺序之间有一一对应的关系，而且元素之间结构关系也须一致。

2）基于更新要求

XML 数据基本形式是 XML 文档。目前，对 XML 数据的处理分为两种不同方式：XML 流处理和静态数据处理方式。静态数据处理方式也就是传统的数据管理形式；而在 XML 流处理方式中，数据的动态性将导致数据需要经常更新，如何有效地实现给定索引动态管理是索引设计时的一种基本考虑。

（1）数据更新有效性。目前对于 XML 数据更新还未形成统一标准。在实际应用考虑中，通常可以给出一些 XML 数据更新操作概念，并就如何在 XQuery 之上实现数据修改功能的扩展进行了探讨。XML 数据通常都会有相应类型限制：当 XML 模式不存在时，XML 须是良好的；当存在一个相关联的 XML 模式时，XML 数据就必须满足模式规定的数据生成规则。由于更新有可能改变 XML 数据结构，那么，需要在更新之前确认这种更新不会改变 XML 模式规定的限制。

（2）索引增量式更新。数据改变将引发索引的相应变动。如果数据只是部分改变，重新建立整个索引将带来很大开销。因此，数据部分改变，索引也应当部分改变，这就是索引的增量式更新。增量式更新一个基本要求是更新后的索引应当与索引的完全式更新效果相同，这就需要维护 XML 数据与 XML 索引之间的一致性，即保证 XML 索引增量式更新能够准确地反映 XML 数据方面的相应更新。

2. 索引基本分类

XML 数据管理领域中关键问题是研究相关支持查询模式的处理技术，而 XML 数据索引技术就是这类支持技术的核心之一。经过多年的研究，已经出现了众多的索引技术。对

于这些索引方法,可以从不同视角进行分类。

(1) 按照索引的数据对象可分为下述情形。

① 基于值对象索引:即在属性值或结点内容上建立索引。

② 基于结点名索引:即在结点标记上建立索引。

③ 基于边或路径索引:即在 XML 文档树的边上建立索引。

(2) 按照索引的应用目标可分为下述情形。

① 简单索引:它包括标记索引、值索引、属性索引等。

② 路径索引:它抽取 XML 数据的结构,索引具有相同路径或标记的结点用于导航查询时缩小搜索的范围。

③ 连接索引:它在元素的编码上建立特定的索引结构来辅助跳过不可能发生连接的结点,从而避免对这些结点的处理可以利用的索引结构包括 B^+ 树、改进的 B^+ 树(R 树和 XR 树等)。

(3) 按照对 XML 数据结构信息处理方式可分为下述情形。

① 结点记录类索引:类似于树结构的保存。基于实际的处理要求,将 XML 数据单元(标签名称、属性名称、属性的值、文本,或者文本中的字符串等)作为记录保存,同时在数据单元的记录中保存有助于确定结点之间结构关系的位置信息。基本模式为(数据单元标识,数据单元在 XML 中的位置信息)。需要存储的位置信息可分为两种情况,一是结点在某种遍历方法所得字符序列中的序号;二是结点在 XML 文档中的路径信息。

② 结构摘要类索引:将 XML 数据按照标签路径进行约简归并,这种约简中只存储保存 XML 数据中不同的标签路径,将具有相同标签路径的结点集合作为约简中标签路径末端结点的内容,此时将常规 XML 数据的路径查询处理转换为通过在约简结构中的处理过程。

在这种分类方式中,相对基于 DOM 和 SAX 的遍历查询,基于结点记录索引可以较好解决"路径多次搜索"问题,而基于结构摘要索引可以较好解决"结点重复访问"问题。以下按照"结点记录"类和"结构摘要"类分别简述相应的具有代表性的 XML 索引方法。

7.5.2 结点记录类索引

如前所述,遍历方法中的第一种问题是由于查询处理必须通过标签路径查找结点这一限制造成的,即要想最终得到满足查询路径的结点,必须顺序地依次访问标签路径对应的结点路径。那么,能否考虑避免必须通过路径查找结点,而在某种辅助信息的帮助下,直接针对结点集合的划分得到最后结果呢?这就形成了结点记录类 XML 索引的基本思想。

结点记录类索引实际上是将 XML 数据分解为数据单元的记录集合,同时在记录中保存该单元在 XML 数据中的位置信息。

位置信息通常分为两类。

(1) 结点在某种遍历方法所得字符序列中的序号信息。

(2) 结点在 XML 文档中的路径信息。

由此按照不同位置信息,就有相应获取信息的两种方法:结点序号法(node numbering method)也称为结点标签方法(node labeling method);结点路径法(node path method)。

这样在实际应用中，根据路径信息不同的表现形式，结点记录类索引分为基于结点序号的索引、基于结点路径信息的索引和二者相结合的混合索引等类型。

应用结点记录类索引查询结果是满足条件结点的集合，按照查询操作的封闭性要求，真正输出的是XML形式，或者至少是具有层次结构的形式。因此，还需要根据相应的结点信息对初步查询结果进行重构，即将适当的结点按照层次结构“连接”起来。

查找给定结点对的某种结构关系的操作称为结构连接运算，这在XML数据处理中也可以看作是一类可以独立存在的基本操作。例如，对于查询表达式a/[b]//c，首先将其分解为a/b和//a//c，再考虑基于父/子关系的连接运算，得到相应结点对(a,b)，同时考虑基于祖先/后裔关系的连接运算，得到相应结点对(a,c)；最后通过适当方式，对得到的两个结点对集合进行装配，从而完成相应查询。

需要指出的是，这种结构连接方法不仅可直接用于XML数据查询过程，即使在应用其他类型索引时，当得到结果是所需查询结果的超集时，为了得到准确查询结果，同样需要进行结点对的连接运算。连接运算的优势在于，当需要考虑祖先/子孙关系时，如a//b，并不需要搜索连接a和b中间“无用”结点，从而可大幅提高查询效率。基于结点记录索引得到的结点带有相应的结构位置信息，通过这种索引技术也是实现连接运算的一种有效途径。

基于结点记录索引的基础是对结点进行“编码”，而编码中带有所需要的结点的结构位置信息，因此结点记录索引也可以看作是结点编码索引。研究一种结点编码方案通常需要考虑下述两个基本因素。

(1) 快速性。能够快速确定XML数据树中任何一个结点对的包含与文档位置关系。

(2) 完全性。能够有效搜索某个特定结构关系在XML数据树中出现的所有情况。

现有结点编码方案大致可以分为以下两种类型。

① 基于路径编码(path-based)。针对结点间嵌套特征，利用绝对路径和其终点的相互确定性，对于每一绝对路径的终点赋予一个编码，这种编码可以是数值型的，也可以是根据结点标签得到的字符型编码。

② 基于区间编码(region-based)。针对结点的有序特征，按照某种遍历顺序，对XML数据树中的每个结点赋予一个编码，这种编码通常都是数组型。

为了表述简便和突出主线，以下假设各类编码方案的数据对象都是XML数据树，即基于有向树模式的XML数据。

1. 基于路径编码

基于路径编码主要有“位向量”编码和“前缀”编码两种方式。

1) 位向量编码

设XML数据树T具有n个结点，则位向量编码是将T中每个结点赋予一个n位向量，当结点u按照某种遍历顺序(如先序遍历)的序号是p，则其位向量编码第p位上的分量就是“1”。

位向量编码是一个由上到下或由下到上的编码方案，每个结点都继承标识其祖先结点或后裔结点的所有位置上的“1”。例如，若结点u的位向量编码为$d(u)=(s_1,s_2,\cdots,s_n)$，当对应模式树上的第$k$个结点为$u$本身或$u$的祖先结点时，$s_k=1$，否则，$s_k=0$。

通过位向量编码，可以有效地确定两个结点之间是否具有祖先/后裔关系。实际上，由两个结点u和v的位向量编码$d(u)$和$d(v)$的合取运算即可确定u是否为v的祖先，u是v

的祖先⇔$d(u) \wedge d(v) = d(u)$；而由析取运算可确定 u 是否为 v 的后裔，u 是 v 的后裔⇔$d(u) \vee d(v) = d(u)$。

由此可知，位向量编码能够有效支持结点间包含关系的检测。位向量编码的一个实例如图 7-6 所示。

2）前缀编码

如果将一个结点的父结点编码作为其自身编码的前缀部分，则该编码就是前缀编码。前缀编码也称为 Dewey 编码。

设 u 是 XML 数据树 T 中的一个结点，其前缀编码为 $d(u)$，则其子结点 v 的前缀编码 $d(v)$ 定义为 $d(v) = d(u).n$，其中 n 为 v 在 u 的所有子结点中序号。

前缀编码具有如下基本性质。

(1) 结点 u 是结点 v 的后裔，$d(v)$ 是 $d(u)$ 的前缀。

(2) 若 u 是 v 的左兄弟结点，则 $d(v)$ 大于 $d(u)$。

由上述可知，前缀编码即可支持包含关系的判定，也可支持文档位置关系的计算。前缀编码的实例如图 7-7 所示。

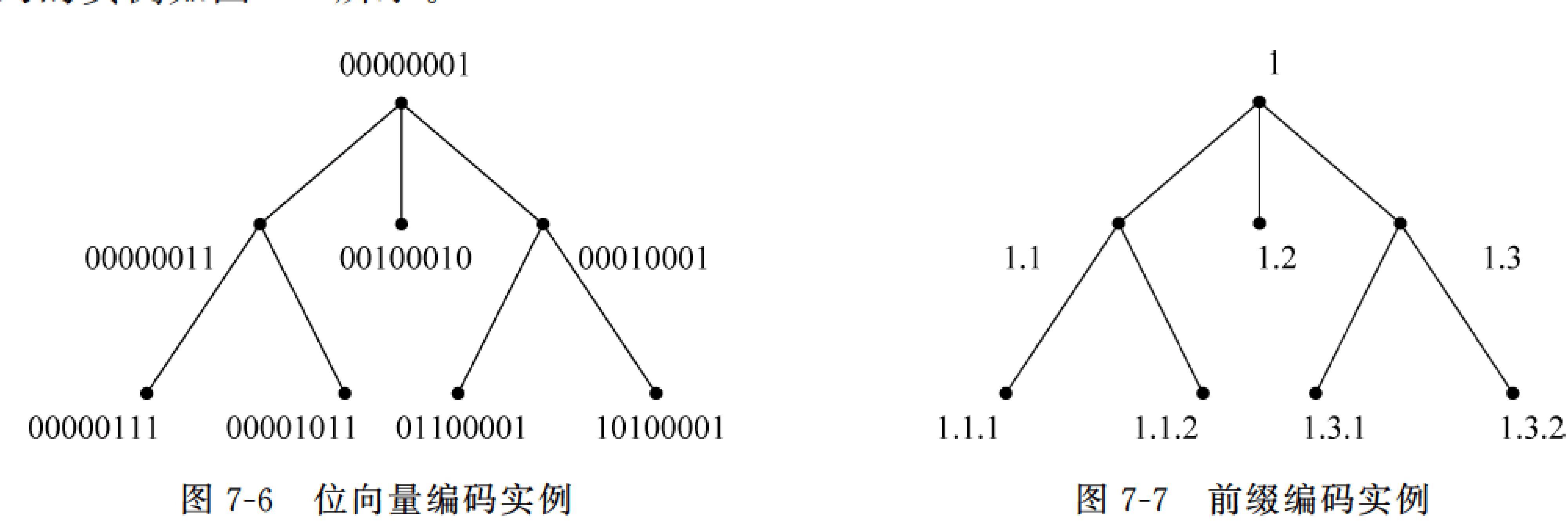


图 7-6　位向量编码实例　　图 7-7　前缀编码实例

2. 基于区间编码

设 XML 数据树为 T。对于 T 中每个结点 u 都赋予一个二元组区间：[begin(u), end(u)]。由两个区间编码包含与否，确定相应两个结点之间是否有祖先/后裔关系。也就是说，区间编码应当满足以下两个条件。

(1) XML 模式树 T 中结点 u 是结点 v 的祖先⇔(begin(u)< begin(u))∧(end(v)< end(u))。

(2) 两个结点的区间(编码)只有包含或相离两种基本关系。

对于区间始点和终点选取不同，可以形成不同类型的区间编码。

1）Dietz 编码

如果将 XML 数据树 T 中结点赋予一个由先序遍历序号为始点，后序遍历序号为终点的区间编码(pre, post)，则编码就构成 XML 树 T 的 Dietz 编码。

XML 数据树中结点 u 和 v 具有祖先/后裔关系⇔(pre(u)< pre(u))∧(post(v)< post(u))。

Dietz 编码的实例如图 7-8 所示。

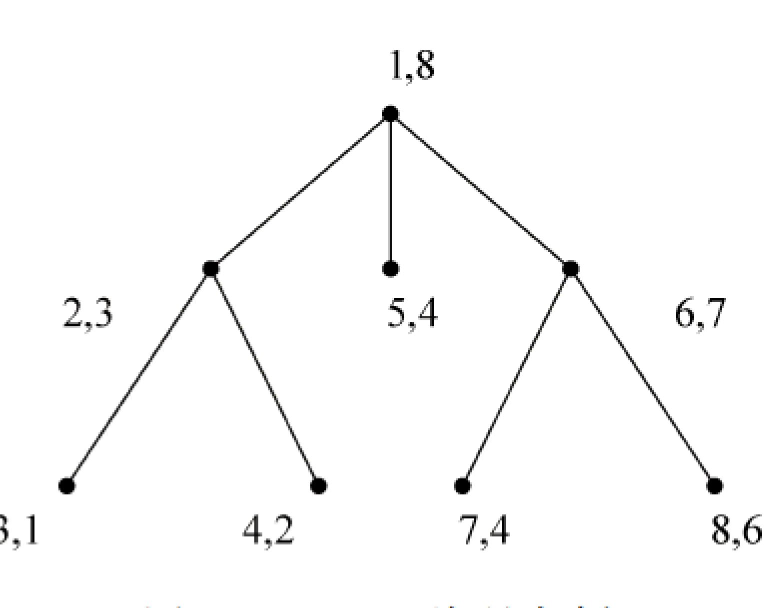


图 7-8　Dietz 编码实例

对于 Dietz 编码方案来说，begin(u)和 end(u)都可以作为结点 u 的唯一标识。

2）Li-moon 编码

在 Li-moon 编码中，区间编码的始点和终点分别为结点的扩展先序编码序号和结点后裔的范围，即 XML 数据树中结点 u 的 Li-moon 编码为（order(u)，size(u)）。作为 u 的扩展先序序号，order(u)为非连续取值，其作用是为结点插入预留相应的序号空间。XML 数据树的 Li-moon 编码（order(u)，size(u)）需要满足下述条件。

（1）若 v 是 u 的父结点，则

$$(\text{order}(v) < \text{order}(u)) \wedge (\text{order}(u) + \text{size}(u)\text{order}(v) + \text{size}(u))$$

（2）若在扩展先序遍历中，x 是 y 的右兄弟结点，则

$$\text{order}(y) + \text{size}(y) < \text{order}(x)$$

（3）若 v 是 u 的所有子结点，则

$$\text{size}(u) \geqslant \sum_{v \in V} \text{size}(v)$$

由上述 Li-moon 编码概念可得到下述基本性质。

（1）两个结点 u 和 v 具有祖先/后裔关系⇔$(\text{order}(u) < \text{order}(v) + \text{size}(v)) \wedge (\text{order}(v) + \text{size}(v) \leqslant \text{order}(u) + \text{size}(u))$。

（2）将结点 u 再赋予一个该结点的深度指标 depth(u)，则结点 u 和 v 满足父/子关系⇔u 和 v 满足祖先/子孙关系且 $\text{depth}(u) = \text{depth}(v) - 1$。

与 Dietz 编码比较，Li-moon 编码能够更好支持数据的更新。

Li-moon 编码的实例如图 7-9 所示。

3）Zhong 编码

对于 XML 数据树 T 进行先序遍历，结点 u 在遍历时被访问两次并产生两个序号，一是在遍历该结点所有后裔结点前访问该结点，由此产生相应区间编码的 begin(u)；二是在遍历完该结点所有后裔结点后再访问该结点，由此产生相应区间编码的 end(u)。由此得到的区间编码称为 Zhong 编码。Zhong 编码具有下述基本性质。

结点 u 和 v 具有祖先/后裔关系⇔$\text{begin}(u) < \text{begin}(v) \wedge \text{end}(v) < \text{end}(u)$在 Zhong 编码中，begin($u$)可以作为结点 u 的唯一标识。

Zhong 编码的实例如图 7-10 所示。

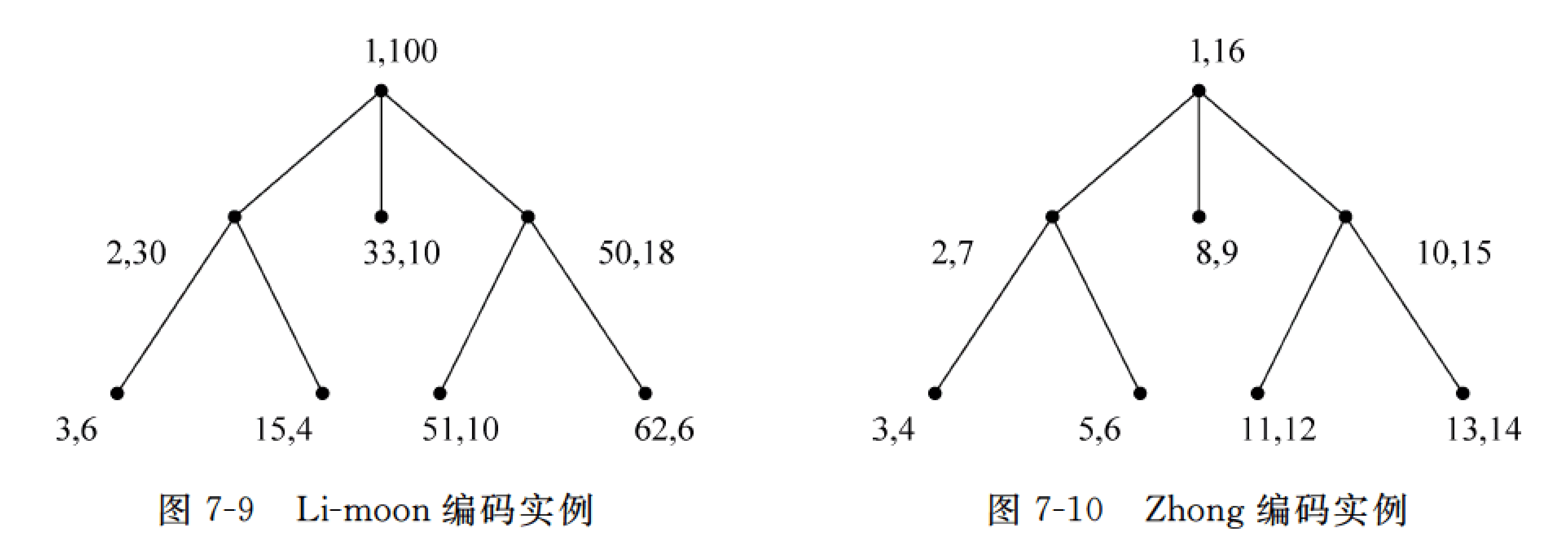


图 7-9　Li-moon 编码实例　　图 7-10　Zhong 编码实例

4）Wan 编码

在 XML 数据树 T 中，将结点 u 的扩展先序序号 order(u)作为区间编码的始点，而将结点 u 的后裔中最大的扩展先序遍历序号 maxOrder(u)作为区间编码的终点，即此时 u 的区间编码为（order(u)，maxOrder(u)），称此区间编码为 Wan 编码。

Wan 编码具有下述基本性质。

结点 u 和 v 具有祖先/后裔关系⇔(order(u)< order(v)) ∧ maxOrder(v)≤maxOrder(u)。

Wan 编码的实例如图 7-11 所示。

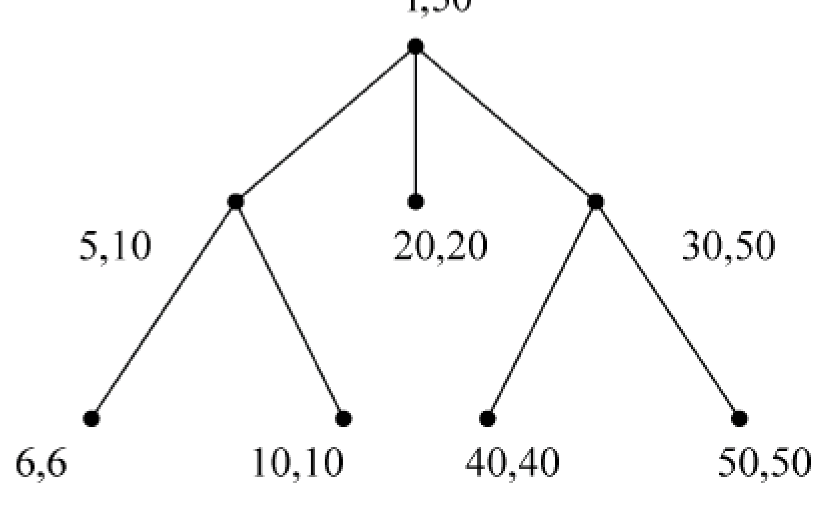


图 7-11　Wan 编码实例

3. 基于 UB-tree 索引

基于 UB-tree XML 索引基本思想：由 XML 数据树 T 中最长路径中包含的边数 d 作为所需多维空间的维数 d，再将 T 中每一条绝对路径映射为 d 维空间中的一个点。这样，一个 XML 数据树就和一个多维空间中的点集相对应，从而可以通过多维空间索引 UB-tree 完成 XML 数据的索引。

在路径索引记录数据的管理中，UB-tree 索引突破了路径匹配只能基于字符串匹配的限制，使用 Z-地址和 Z-区间，通过对路径的转换，实现了借助于转换对路径的多维管理。

Z-地址：设 Ω 表示 n 维空间，Ω 空间中每条记录的第 i 个属性 A_i 具有 s 个可能的取值，表示为 $A_i = A_{i,s-1}, A_{i,s-2} \wedge A_{i,0}$，对 Ω 空间中的任一记录 $o \in \Omega$，定义唯一的 Z-地址函数 $Z(o)$ 与之对应：

$$Z(o) = \sum_{j=0}^{s-1} \sum_{i=1}^{n} A_{ij} \; 2^{j+i-1}$$

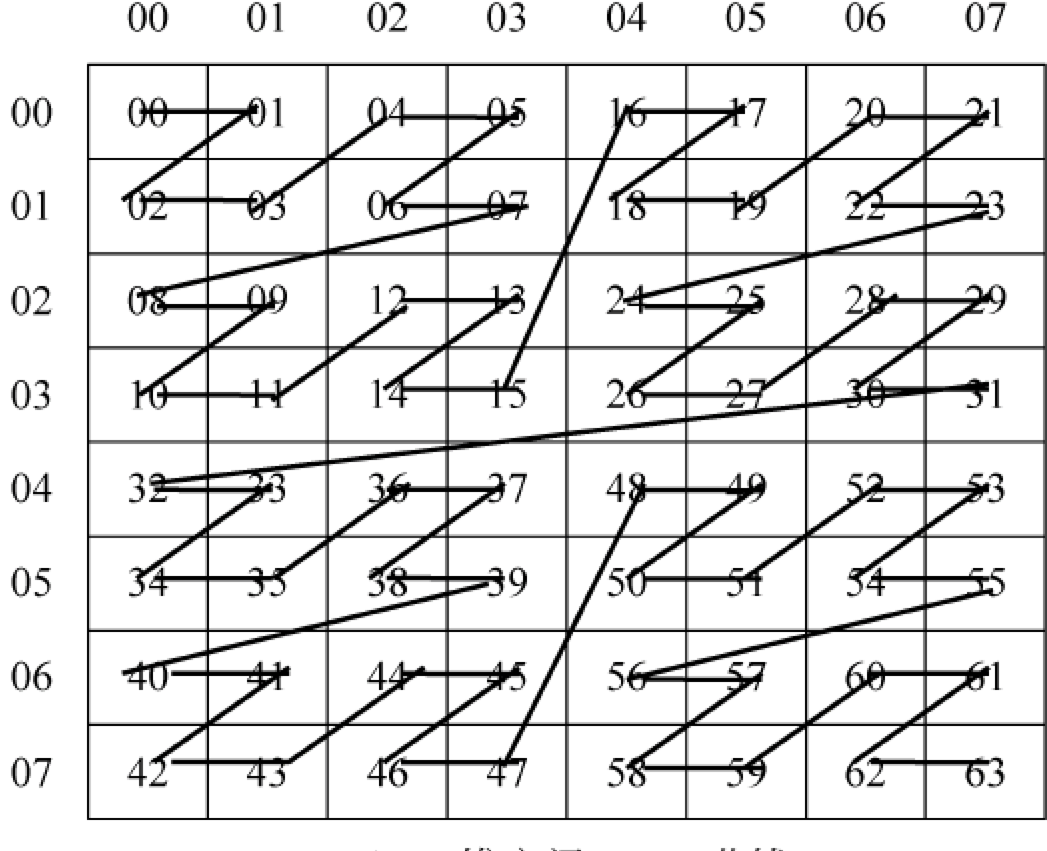
利用 Z-地址，可以构建一个从 n 维空间到一维 Z—地址空间的映射。n 维空间中每一点都对应 Z—地址空间的一个区间(α,β)，称为 Z-区间(Z-region)。以 B^+ 树对 Z-区间集合中的数据构建索引，就成为 UB-tree 的基本考量。当将 XML 路径集合看作是某个 n 维空间(取 XML 数据中最长路径的长度作为维度 n 的值)中的一个实例时，就可以实现将 XML 路径到 Z-地址空间的转换，进而可以实现用 UB-tree 来对 XML 路径信息进行索引的目的。

二维空间中点的 Z-地址和由点组成的 Z 曲线如图 7-12 所示。

	00	01	02	03	04	05	06	07
00	00	01	04	05	16	17	20	21
01	02	03	06	07	18	19	22	23
02	08	09	12	13	24	25	28	29
03	10	11	14	15	26	27	30	31
04	32	33	36	37	48	49	52	53
05	34	35	38	39	50	51	54	55
06	40	41	44	45	56	57	60	61
07	42	43	46	47	58	59	62	63



(a) 二维空间8×8 Z-地址



(b) 二维空间8×8 Z曲线

图 7-12　Z-地址和 Z-曲线

虽然这种基于转换的多维管理结构在转换代价和路径转换空间冗余两个方面存在不足，但由于多维索引的概念能够突破对字符串只能进行连续模式匹配的方式，因此，在这个

方向上做进一步的研究具有较好的实际意义。

7.5.3 结构摘要类索引

如前所述,当 XML 有向树中还需要添加“引用边”时,就需要通过 XML 具根有向图来表示相应的 XML 数据模式。结构摘要类索引适合于更一般的 XML 具根有向图情况。本节索引的数据对象称为 XML 数据图。

由 XML 构成概念可知,XML 数据图中可能有不同的数据结点具有相同的语义标签,由此就可能存在着不同的数据结点具有相同的绝对或相对路径表达式。XML 查询表达式基于 XPath,而 XPath 中的路径表达式实际上是标签路径表达式。如前所述,一个标签路径表达式可以包含多个不同的结点路径表达式,但通常需要得到基于结点路径的结点查询结果,由此就会出现前述表明的“多次路径搜索”问题。

结构摘要类索引基本思想:以 XML 数据图结构中结点的路径信息为基础,采取适当某种约简方式对图中的标签路径进行约简归并,使得约简后的数据结构只需维护不同的标签路径信息,而不会出现具有相同标签路径的两个不同结点。

设 XML 数据(有向)图表示为四元组 $G=\left(\mathrm{VG},\mathrm{EG},\mathrm{rootG},\sum G\right)$,R 是 VG 上的等价关系。下面给出基于 R 的结构摘要索引的形式化定义。

G 基于 R 结构摘要索引模式表示为一个标签有向图 $\mathrm{IG}=\left(\mathrm{V_I(G)},\mathrm{E_I(G)},\mathrm{root_I(G)},\sum G\right)$

(1) $\mathrm{V_I(G)}$:IG 中结点集合,是 VG 中满足等价关系 R 的结点等价类,称为索引结点,并记作 u. extent。其中 u. extent 表示 VG 中结点 u 基于等价关系 R 的等价类。

(2) $\mathrm{E_I(G)}$:IG 中边的集合,其中边元素满足下述条件:

$\forall g\ (u_i,v_i)\in \mathrm{E_I(G)}\Leftrightarrow \exists\,(u_d,v_d)\in \mathrm{EG}$,满足:$u_d\in u_i.\mathrm{extent}$,$v_d\in v_i.\mathrm{extent}$

在结构摘要索引中,最初多采用自动机理论中的 NFA 到 DFA 转换的思路。稍后,引入能够准确地表述结构摘要的双拟概念问题,现有工作多是基于这个概念。

结构摘要对于具有相同标签路径的文本数据都只保留了唯一的标签路径,具有相同标签路径的文本数据都集中在该路径结点之中。这样,只要搜索一条标签路径就可以获取相同标签路径的所有结点。

(1) 按照适用的查询路径划分,现有结构摘要索引可分为两类:基于线性路径查询索引和基于分枝路径查询索引。

线性路径查询更为基本,分枝路径查询比较复杂。基于分支路径查询主要有覆盖索引、F&Bindex 和 Index Fabric 等。

(2) 按照索引是否支持数据更新可分为以下两种类型。

① 不支持更新的索引:例如,在线性路径查询情况下,其主要代表为 DataGuides、1-index 和 A(k)-index 等,它们都是一种静态索引,如果数据源发生改变,索引更新代价通常较高。

② 支持更新的索引结构:例如,在线性路径查询情况下,其主要代表为 APEX、D(k)-index 和 M(k)-index 等,这些都是所谓的动态索引,即索引结构设计都考虑数据更新因素,具有某种能够根据数据源变化“自动”修改索引结构的功能,也就是具有更新的“自适应性”。

以下介绍基于线性路径查询的结构摘要类索引中的代表性方法,这主要是 DataGuides、1-index、A(k)-index、APEX 和 D(k)-index。其中,DataGuides 通常被认为是较早出现的结构摘

要类索引,它能够精确记录 XML 树中出现的所有路径,对于自根向下的路径查询非常有用,但是不提供任意两个结点的关系的判断,不能从任意结点开始进行向下的路径查询。1-index 和 A(k)-index 通过对结构摘要等概念的形式化和精确化方式,进一步改进了 DataGuides 中的方法。APEX 和 D(k)-index 分别是基于更新的具有自适应特性的结构类索引。

1. DataGuides

作为结构摘要类索引的较早期代表,DataGuides 索引已在 Lore DBMS 中得到了实现。设 Gxml 表示 XML 数据图,Gind 表示基于 DataGuides 的 XML 数据索引图,则 DataGuides 的基本思想是对于 Gxml 中的任何一条标记路径,在 Gind 中存在且只存在一条完全相同的标记路径;而对于 Gind 中的任何一条标记路径,在 Gxml 中也一定存在相同的一条标记路径。图 7-13(b)和图 7-13(c)都是图 7-13(a)的 DataGuides 索引。

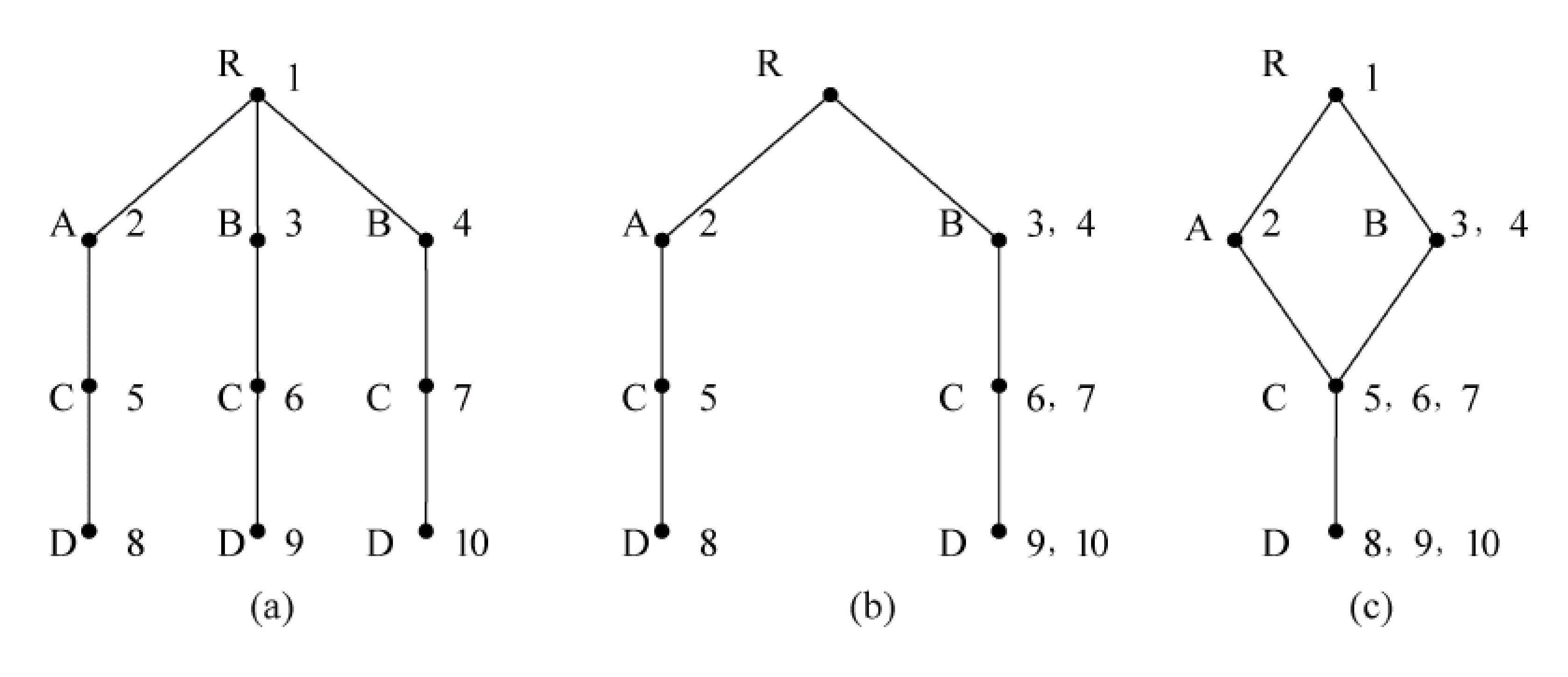


图 7-13　XML 数据和 DataGuides 索引

从图 7-13 可知,一个 XML 数据树的 DataGuides 索引并不唯一,如图 7-13(b)和图 7-13(c)都是图 7-13(a)的 DataGuides 索引,都能起到数据索引的作用,而图 7-13(c)显得更为简洁,直观上似乎应该选择图 7-13(c)。但图 7-13(c)作为索引使用实际上并不合适。在图 7-13(c)中,考虑到图 7-13(a),索引结点 A 对应于数据结点 5、6、7,索引结点 D 对应于数据结点 8、9、10,因此,当查询路径为 R/A/C/D 时,得到的查询结果集合就为{8,9,10}。实际上,通过这一条路径,并不能够获取结点 9、10。同时,索引的维护(插入、删除等)是衡量一个索引品质的重要因素。由于这种索引构建的不唯一性,对图 7-13(a)的任何更新都可能给索引带来各种影响,付出令人望而生畏的更新开销。

DataGuides 索引不具唯一性的原因在于其中标签结点的“归并”标准不够明确,即 DataGuides 中并未就“归并”或“结点等价”等基础概念做出准确描述。事实上,如果 DataGuides 中索引结点中数据结点存在不等价现象,则该索引就可能并不适用。为了解决这个问题,需要进一步引入数据结点的“进入路径”概念。

进入路径:XML 数据图 G 中一个数据结点 u 的进入路径(incoming path)是一条由当前结点到 u 的(标签)路径。

需要注意的是,在数据图模式中,由当前结点到达一个确定结点 u 可能存在多条当前路径,因此明确结点进入路径非常必要。基于进入路径的概念,人们提出了 srongDataGuides 索引,该索引的基本点在于当两个数据结点存在至少一条相同进入路径时,方可认定 u 和 v 是可以归并的即“等价”的。图 7-14(b)是图 7-14(a)的 StrongDataGuides 索引。

DataGuides 与 StrongDataGuides 索引的不同在于路径合并的策略。

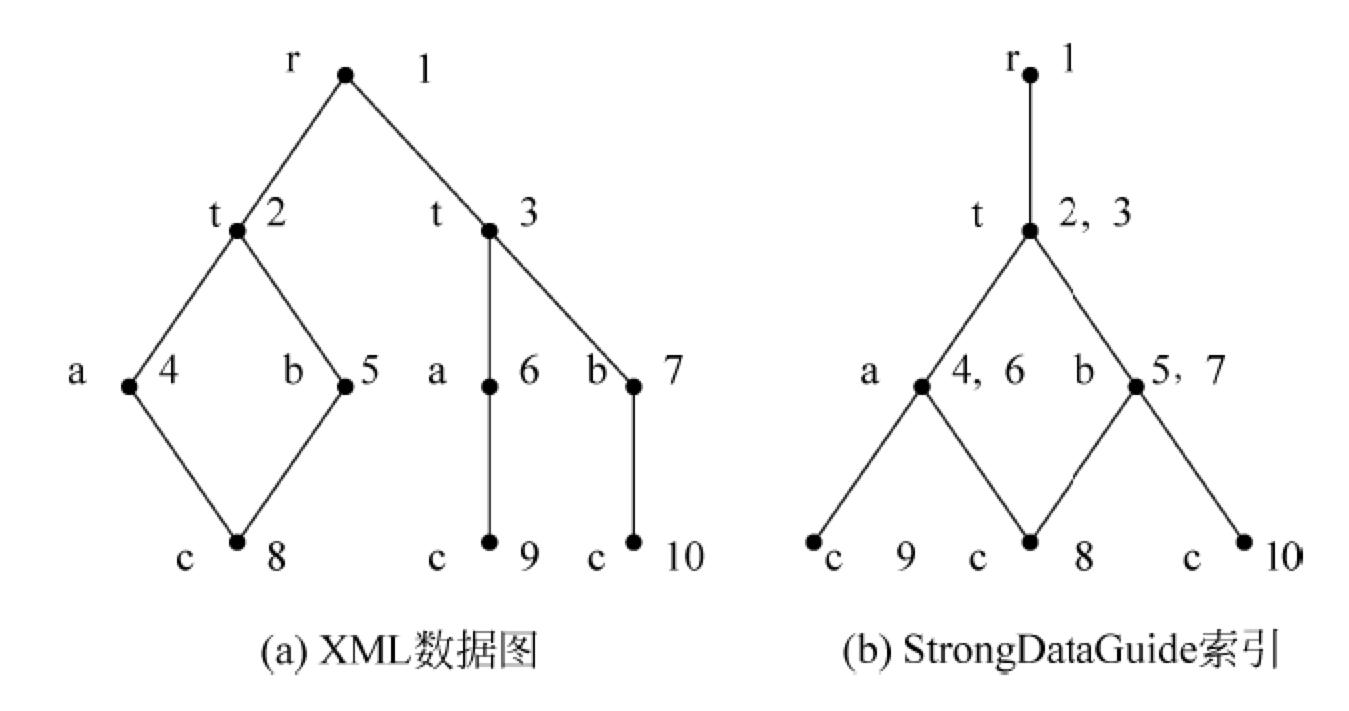


图 7-14　XML 数据图和 StrongDataGuides 索引

2. 1-index

由于 DataGuides 索引的核心是标签结点的归并，而这种归并还要保留数据结点间原有的结构信息，因此需要将“归并”标准明确化和规范化。StrongDataGuides 对此做了初步尝试，但是还可以进一步深入。这就是 XML 数据图中两个数据结点 u、v 的双拟概念。

在 XML 数据图 G 中，当两结点 u 和 v 满足下述条件，则称 u 和 v 是双拟的(bisimulation)。

(1) 当 u 是根结点时，v 也是根结点。

(2) 当 u 和 v 是非根结点时，两者具有相同标签，并且如果结点 u 和 u_0 之间存在连接边(u,u_0)，v 和 v_0 之间也存在连接边(v,v_0)，而 u_0 和 v_0 也是双拟的，同时，反之亦然。

如果 XML 数据图 G 中两个结点 u 和 v 是双拟的，则称两个结点 u 和 v 之间存在双似关系(bisimilarity)，并记为 $u \approx v$。

双似关系是满足自反性、对称性和传递性的等价关系。结合“进入路径”概念可知，两个结点 u、v 具有双似关系$\Leftrightarrow u$ 和 v 的所有进入路径集合相同。

需要注意的是，在 StrongDataGuides 中，只要两个元素有一条进入路径相同，则认为这两个元素可以归并。如果以双似关系作为结点归并条件，即当 u、v 具有双似关系时，u 和 v 才能归并，那么实际上就得到比 StrongDataGuides 更强的归并条件：结点 u、v 可归并$\Leftrightarrow u$ 和 v 所有的进入路径都相同。以此为结点归并条件得到的结构摘要索引就是 1-index。

1-index 的实例如图 7-15 所示。

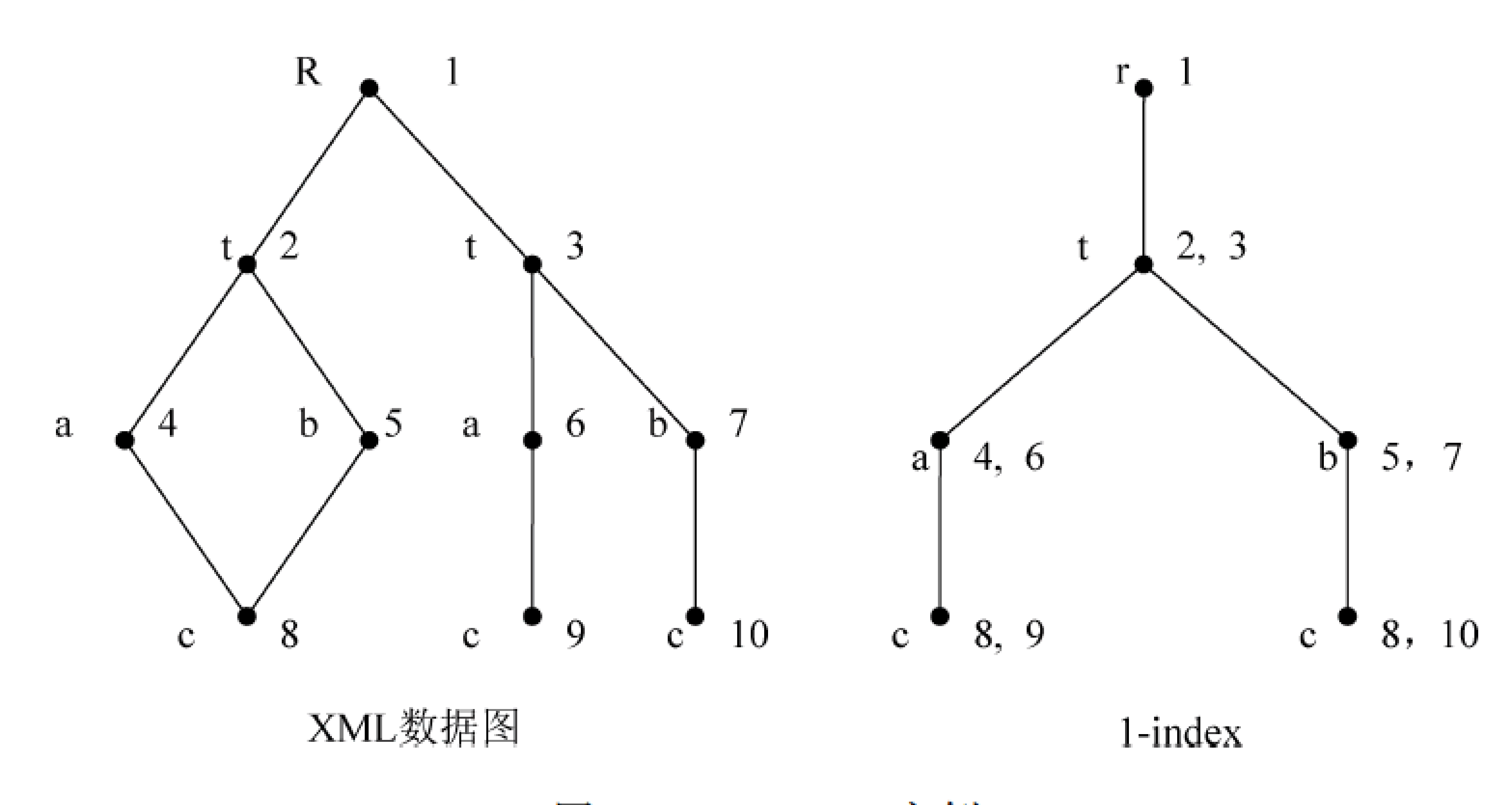


图 7-15　1-index 实例

3. A(k)-index

StrongDataGuides 和 1-index 中采用的进入路径都是基于绝对路径，由此构建的索引能够得到可靠准确的查询结果。但在实际应用中，需要查询的内容通常都是 XML 数据片段，这需要通过相对路径的方式实现，上述两种索引中的等价划分显得过于严格和拘谨。因此，还需要考虑两个结点元素之间的局部等价概念以提高查询效率。为此引入 k 阶双似概念。

两个结点 u 和 v 称为 k 阶双似，并记为 $u \approx^k v$，如果满足如下条件。

(1) $u \approx^0 v$ 当且仅当 u 和 v 具有相同标签。

(2) $u \approx^k v$ 当且仅当 $u \approx^{k-1} v$，同时，u、v 的所有父结点 u_0 和 v_0 也满足 $u_0 \approx^{k-1} v_0$

实际上，如果结点 u 和 v 是 k 阶双似的，则表示 u 和 v 的长度为 k 的进入路径集合相同，从而将全局进入路径集合扩展到局部进入路径集合，由此产生的索引结构对于某些常用的局部查询是相当有效的。

A(k)-index 实例如图 7-16 所示。

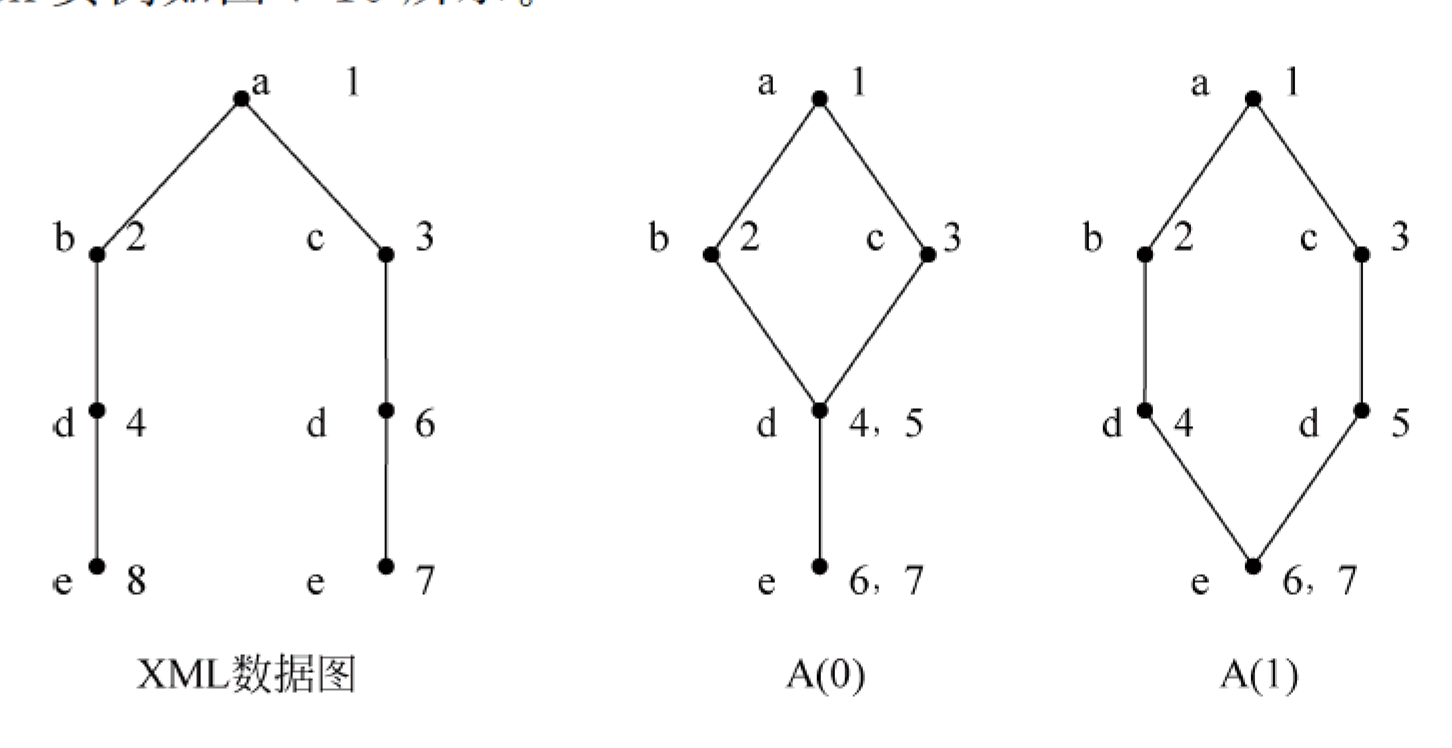


图 7-16　A(k)-index 实例

4. APEX

实际应用中，数据源中的数据会经常发生变动，对于网络数据尤其如此，因此，XML 数据相应的索引结构也需要发生改变。从更新效率考虑，XML 中不同结构片段在实际查询过程中出现的频率不同。针对那些经常查询的 XML 数据片段或路径进行重点关注和优先处理，从而使得相应的结构更加精细和准确；而对于相对出现不频繁的部分可以一般处理，使得在索引能够较为方便容易地实施更新。这种具有"自适应"能力的索引结构可更为有效地完成相应的数据查询和数据更新。最早体现出这种自适应考虑的 XML 索引可能就是 APEX 索引(Adaptive Path Index for XML)。

(1) APEX 基本组成。一个图形结构和 Hash 表。

图结构主要保存长度为 2 的所有路径和某些在具体应用中经常出现的路径。Hash 表将某些查询路径与索引中结点相互对应。

在 APEX 中，对于已保存的路径，在查询时只需要在 Hash 表中完成查找。另外，Hash 表还保存有结构摘要中所有结点集的入口，对于带有"//"的部分匹配，并不需要进行遍历查找，从而提高了部分匹配的查找效率。

(2) APEX 查询过程。首先，确定查询表达式中是否有已经保存的"频繁"路径。如果存在，则直接在 Hash 表中找到查询入口。此时，如果查询长度为 2，则就由图结构中搜寻精确答案。其次，如果查询长度大于 2，则对图结构中某些相关结点结合进行连接或交运算，

并记录相应查询路径,调整频繁路径集合,然后根据频繁路径集合调整整个 APEX 索引。对于完全非频繁路径查询,也可仿照上述处理。APEX 的基础是频繁路径集合确定,这通常是采用数据挖掘技术由查询负载中获取。

5. D(k)-index

作为一种自适应索引结构,D(k)-index 是建立在 A(k)-index 基础之上的。

D(k)-index:根据 k 阶双似关系概念进行结点的归并聚类,但并不事先给定一个应用于全局的 k 值,不同等价类可以根据不同的 k 值进行聚类。对于频繁和重要的等价类,取相应的较高的 k 值,以提高对相似度的要求;对较不频繁和次要的等价类,相应 k 值可取得较小。这里等价类的重要和频繁与否,通常根据实际应用确定。

这种根据各个等价类确定的 k 值称为等价类中元素的局部相似度。

一般而言,具体结点的局部相似度可以根据下述两个查询路径和结点结构确定。

(1) 查询路径:对于长度为 p 的查询路径 n_1、…、n_p,使得对于路径中出现的每个结点,$k(n_i) \geqslant p$,对于路径中不出现的结点(如在"//"情形),其相似度 $k=0$。

(2) 结点结构:如果存在由结点 n_0 到 n_1 的连接边(n_0, n_1),则 $k(n_0) \geqslant k(n_1)$。

D(k)-index 相对于 A(k)-index 降低了索引规模,但也还存在无关索引结点过于细化问题。无关数据结点过度细化、父结点相似度过高引起的结点过度细化和短路径查询效率不高等都是 D(k)-index 的不足。M(k)-index 和 M*(k)-index 就是针对上述问题对 D(k)所做的改进,这里就不再予以讨论。

本章小结

文本数据只能进行基于文件名的查询,输出结果为整个文件,也就是说,查询不能深入到文件的内部语义内容。而数据库文件的查询能够有效地深入到相应数据文件内部选取所需要的内容,甚至还能将多个数据文件中的个别内容进行整合(如 RDB 中连接查询等)作为查询结果输出。作为一种语义标记语言,XML 的意义就在于能够将文档解构为各个具有"语义"的单元,从而使得相应查询也能深入相应文本内部进行有效的语义内容抽取。由此可见,XML 的查询处理实际上可以看作是隐含了进行数据库意义下数据查询的基础,这样,讨论 XML 数据管理技术就是一种自然而然的事情了。

XML 以文档形式出现,但由于其中出现的"元素"具有嵌套特征等而不同于一般文档文件。XML 是一种半结构化数据,其特点之一就是描述数据模式的"元数据"和"实际数据"相同的形式,可以只用一般的半结构化数据模型予以描述,即为其构建相应的基于有向树或具根有向图的数据模型,从而为 XML 数据库研发提供了模型方面的基础支撑。

由于半结构化数据的数据模式可以从已有数据实例抽取,同时并不要求所有数据实例都具有显式的数据模式描述,因此就需要区分满足一定模式结构要求的 XML 文档和不满足模式要求甚至没有模式结构要求的 XML 文档。数据库管理数据的一个基本特征就是需要在逻辑层面存在基于相应数据模型的数据模式,并以此来对各种各类的数据实例进行整体的统一管理,因此,从数据库角度而言,需对 XML 数据也构建起具有数据模式与数据实例意义的数据关联,而 XML 标准中的 DTD 和 XML Schema 就起到了数据模式的作用,而满足它们规范要求好制约的 XML 文档相当于相应的数据实例。XML 文档可以使用 DTD

或者 Schema 来表示该文档具有哪些元素,元素之间具有怎样的层次(嵌套),每个元素具有怎样的属性等。DTD 具有较多局限性,Schema 本身呈现 XML 文档风格,具有更强表现能力,但相对比较复杂。符合 XML 语法规范的称为良好 XML 文档,符合 DTD 或 Schema 模式要求的称为有效的 XML 文档。

看待 XML 数据管理有两种认知角度。一种是将 XML 看作文档,作为文件系统进行管理,这主要适用于难以提取或者没有必要提取统一 XML 模式的一般情形,如人们浏览的网页等。另一种是将 XML 看作半结构化数据,这主要是一些 XML 模式相对固定而且数据量巨大的一些场合,如人事档案管理和医疗数据管理等。

结构化数据是将数据内容和数据模型分离开来进行研究和处理,如关系数据库中的数据表数据和元数据。半结构化数据是将数据内容和数据模式融合一体,统一进行描述和处理。结构化数据有利于数据操作的设计与实现,半结构化数据在应用中则更加彰显其灵活性。从数据观点出发,XML 就是半结构化数据的典型代表,可以按照一般半结构化数据模型 OEM 进行建模,但相应的数据查询较为复杂。

将 XML 建模为半结构化数据的树形或图形结构具有两方面的意义。

1) 依据树或图的性质进行遍历查询

此时主要的查询语言有 XPath、XSLT 和 XQuery。其中,XPath 是路径表达式的一种新标准,允许使用类似于文件系统中路表达式来指定所需要的元素;XSLT 提供了强大的数据查询和样式表转换功能,应用相当广泛;XQuery 是 W3C 开发的 XML 查询语言,其风格与 SQL 相近,可以用于 XML 文档,也可以用于 XML 数据库查询。有人预言 XQuery 将可能成为 XML 数据库中 SQL。

XSL、XSLT、XPath 和 XQuery 之间的关系如图 7-17 所示。

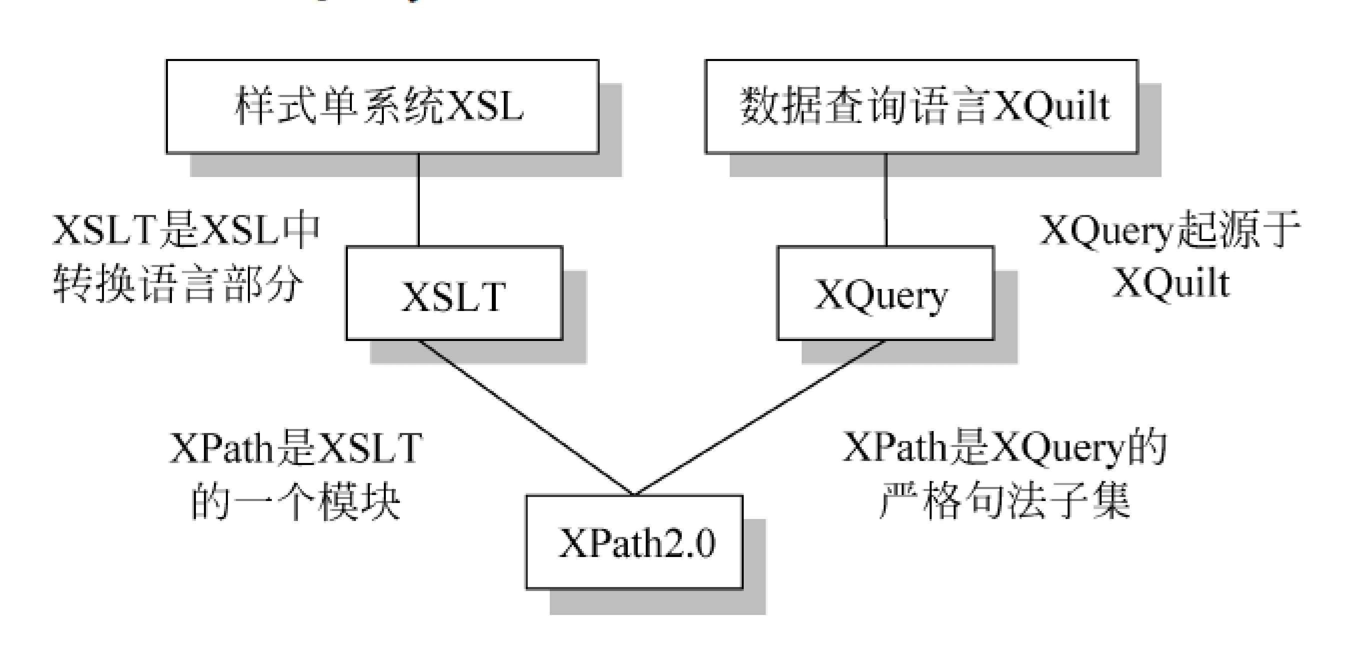


图 7-17　XSL、XSLT、XPath 和 XQuery 之间的关系

2) 基于索引技术的非遍历查询

XPath 和 XQuery 可以通过 DOM 和 SAX 对 XML 文档进行遍历查询,但更为有效的存储与查询是通过 XML 数据索引的途径。XML 查询需要得到的信息较多,但最主要的是结构信息查询。查询表达式基于 XPath 的标签路径表达式,但查询结果实际上是需要满足祖先/后裔等关系的结点,这就需要对结点路径表达式进行搜索。一条标签路径表达式可以对应多条不同的结点路径表达式,这就提供了对标签路径表达式进行约简归并以建立结点摘要类索引的可能性,由此可以建立起 XML 结构摘要类索引。XML 数据树(图)中的结点没有常规意义下的顺序,但可以进行某种遍历以建立适当顺序,这种排序需要包含结点的位置与相互关系信息,基于这种排序就可以赋予每个结点一个编码。对于 XML 数据树(图)

对应的编码集合,可以按照 B^+ 树或其他经典方式进行索引,由此可以建立起 XML 结点记录类索引。数据索引是 XML 数据库技术中的重要课题之一。

XML 数据库是在 XML 数据处理基础上发展起来的。XML 数据库有两个研究途径,一是通过研究 XML 数据与传统数据(关系数据或对象数据)之间的相互映射关系,拓展关系数据库系统或面向对象数据库的相应功能,使其能够进行 XML 数据的管理与查询,这就是使能 XML 数据库;二是纯粹由 XML 本身具有状况出发,充分考虑到 XML 数据特点,以一种自然畅顺的风格与方式处理 XML 数据,从各个方面较好地支持 XML 数据的存储与查询,这就是原生 XML 数据库。

主要参考文献

[1] Akmal B Chaudhri,Awais Rashid,Roberto Zicari. XML 数据管理[M]. 邢春晓,张志强,等译. 北京:清华大学出版社 2007.
[2] 孟小峰. XML 数据管理概念与技术[M]. 北京:清华大学出版社,2009.
[3] 万常选,刘喜平. XML 数据库技术[M]. 2 版. 北京:清华大学出版社,2008.
[4] 萨师煊,王珊. 数据库系统概论[M]. 5 版. 北京:高等教育出版社,2016.
[5] 孔令波,唐世渭,杨冬青,等. XML 数据索引技术[M]. 软件学报,2005,16(12):1000-9825.

第8章　移动对象数据库

进入新世纪以来，无线通信技术、卫星全球定位系统和无线互联网技术得到了快速的发展与普及，跟踪、存储和提供移动对象的位置信息服务成为可能。同时计算机技术自身飞速发展和计算机应用的不断扩展带来计算设备的小型化和便携装置的普及化，随之而来的是计算技术从静止的单一桌面框架走向万千的移动用户，信息交互设备从专业应用走向了普通人的日常生活。其中，移动对象的连续运动对数据库技术提出了新的需求和挑战，并迅速成为新世纪数据库领域的一个研究热点。实际上，在当今世界，人们迫切需要能在任何时间和任何地点，通过任何方式来接收、查询和处理相关数据。无线传感器网络和定位技术（如GPS）的快速发展，众多具有定位功能的无线手持设备和车载设备快速普及，使得许多新的应用产生大量的运动数据信息，这些随时间变化的移动位置数据需要有相应的数据库系统进行有效管理。在传统数据库中，如果数据没有显式更新，数据属性值就会保持恒定，因此难以高效地管理通常在后台连续变化的动态位置信息，由此移动对象数据库（Moving Objects Database，MOD）技术也就应运而生并得到不断发展，其目标是在数据库中高效地管理移动对象的位置及其相关数据。MOD是数据库技术的扩展，用于在数据库中支持移动对象位置信息的存储和管理。

8.1　MOD概述

MOD研究起源于20世纪90年代中期。移动对象随着时间演进而产生地理空间位置上的变化，从逻辑上来看，MOD属于时空数据库的范畴，也可认为来源于描述地理空间数据的SDB和处理随时间变化数据的TDB的整合。移动对象的特点是其位置信息数据随时间演进不断连续改变，因此MOD的基本点就是关注移动点地理空间位置随时间的连续变化，而早期时空数据库只支持地理空间位置的离散变化。事实上，当前MOD技术已经成为时空数据库重要研究课题之一，Erwic等人甚至直接认为时空数据库的本质就是关于移动对象的数据库。

8.1.1　移动对象数据

所有客观事物都是发展变化的，但由于认识论和技术处理的层次递进性，人们研究其性质和展开其应用通常分为两步。首先是将其“看作”静态的，即只考虑“当时当地”的“快照”；其次再继续深入扩展，将所有“快照”整合处理，将其作为一个发展演变的整体进行研究。着眼于事务特性的关系数据和着眼于空间属性的空间数据都是基于“静态”的情形，如关系数据随时间变化就是前述的时态（关系）数据，空间数据随时间发生变化就是时空数据，这都是

一种基于客观事物"动态"的情形。静态是动态的基础,动态是静态的深化,因此时态数据和时空数据管理就构成高级数据库技术的基本内容之一。

如前所述,空间对象可分为只有位置的"点"对象和既有位置又有形状的"区域"对象两种情形。着眼于随着时间演进其位置与形状变化的空间对象就是时空数据对象,研究时空对象数据管理的就是时空数据库。

空间数据的基本特征是其位置和形状,动态空间对象是其位置和形状分别或同时都随时间演进而发生变化。位置的变化就是对象的"运动"或"移动"。具有"位置"变化的空间对象通常称为移动对象,或者说,着眼于位置变化的时空对象就是移动对象,以移动对象数据为研究主体的时空数据库就是 MOD。时空数据库是 SDB 与 TDB 的整合,而从理论渊源上考虑,MOD 是时空数据库的特例。

1. 移动点对象与区域对象

作为位置不断随着时间变化的空间对象,移动对象的变化可以是离散的,也可以是连续的。从几何特征考虑,移动对象可以分为两大类:移动点对象与移动区域对象。

(1) 移动点对象:随时间演进只考虑其地理位置信息变化的客观实体。

例如,行驶中的汽车、飞机和舰船,位置不断发生改变的移动手机用户、具有无线通信功能的笔记本电脑,在交通线上运行的出租车和救护车、敌方来袭过程中的巡航导弹和轰炸机群及逃跑过程中的恐怖分子等。这类移动对象的基本特征是人们只关注对象相对于某一坐标原点的地理位置,即移动点对象是对客观实体在空间中地理位置信息的抽象。

(2) 移动区域对象:随时间演进其地理位置和几何形状都发生变化的客观实体。

例如,运动中的云层和风暴、迁徙中的动物种群和不断推进的沙漠、变动中的冰川和植被区域、燃烧中的森林火灾范围和传播中的疾病区域等。移动区域对象的基本特征是着眼于空间对象位置变化的同时还必须考虑其形状的变化,即移动区域对象是对客观实体在空间中地理位置和几何形状信息的整合抽象,描述实体的移动、扩张和收缩。

从数据管理角度而言,空间对象形状描述相当困难,同时考虑空间对象的位置与形状变化则更具挑战性,当前 MOD 主要研究的是移动点对象,本章中所涉及移动对象通常都是指移动点对象。

2. 移动对象数据特征

相对于常规数据,移动对象数据具有下述基本特点。

(1) 多样性与复杂性。通过 SDB 技术已经知道,空间对象表示的信息特别是其几何位置信息呈现出多样化特点,而空间对象之间关系信息包括位置关系、序关系和拓扑关系等。同时,在 TDB 技术中,时间标签(主要是时间期间)之间也呈现出较为复杂的情形。作为一种特定时空对象,移动对象数据实际上可看作空间数据和时态数据的整合,同时移动对象还需要具体处理连续变动的位置信息,因此,其表示的内容自然更加多样,彼此间时空关系的描述会更为复杂。

(2) 移动对象随机性。移动对象的移动具有随机性和规律性的双重特征。例如,出租汽车根据乘客需要到达指定地点,而不同乘客要去的地点都有所不同,所以从整体上来看,当出租车在运营中并不总是能确定出租车的下一站要驶向哪里,这就表现为随机性。另外,出租车虽然不确切知道下一步要去哪里,但根据交通需求的特点,一段时间(如数周、数月)出租车的行驶轨迹可能存在某种程度的统计规律性,这也是交通预测和规划等的重要依据。

（3）静止和运动交织性。任何移动对象都具有一个起点和终点，都具有静止和运动两种基本的状态。例如，车辆运行过程中出现交通拥堵或车辆故障等，这些都是运动过程中可能出现的静止情形。随着运行过程中静止与运动的随机交替，移动对象的运动速度、方向和行驶轨迹等运动状态也会随机变化。

（4）不精确和不确定性。首先，移动对象运动是连续的，计算机却只能按照离散方式进行数据的存储和计算，因此需要进行采样。由于采集设备精度的限制，移动对象存储的位置信息在空间上与实际的位置上会存在一定的偏差。其次，信息传输的延时和通信处理的开销等影响，移动对象存储的具体信息与实际的位置在时间上总是有滞后。另外，如上所述，移动对象的运动具有各种不可预测的随机性。最后，移动对象不同于传统数据库的突出特征是可以管理当前与未来位置信息，这在本质上是一种基于预测技术的管理，预测必然会伴随着不精确性与不确定性。因此，不管采用哪种“精确度”的位置管理及位置信息更新策略，由于移动对象本身特征所规定，移动对象数据库中保存的位置信息与移动对象的实际位置总会存在一定的差异，在逻辑上都表现为某种程度的不精确性和不确定性。

8.1.2 数据类型和数据管理

随着计算机技术、Internet 技术、无线通信移动定位和 GIS（Geographic Information System）技术的进一步发展，通过由移动通信和互联网融合的移动环球网（mobile web），用户可以在任何时间和地点获得移动用户所在位置的相关服务，能够实现对移动对象的定位及追踪，即定位服务（Location Based Service，LBS），由此产生了大量移动对象数据需要进行存储和管理。MOD 正是在这样的应用驱动中产生和发展起来的。MOD 记录和管理移动对象在不同时刻的位置信息，用户可以在数据库中查询移动对象的过去、现在与未来时刻的状态信息。

1. 移动对象数据类型与存储

如前所述，在 MOD 中，移动对象通常都是移动点对象，而移动对象数据也是指移动点对象的相关位置数据信息。这种位置数据信息从逻辑上可以分为空间位置点和空间运动轨迹两种基本类型，它们都可以借助于关系表或其他方式进行有效存储。

1）位置点数据类型及存储

空间点的位置数据通过对移动对象在移动过程中的采样获取，每个位置点数据包括 X 及 Y 坐标、运动方向、速度和时间等信息元素。这些信息可作为一个元组按照采样顺序依次存储在 RDB 中。由此 MOD 存储了一个移动对象的所有位置点序列数据。借助于 RDB 对原始数据进行存储，灵活性好，可满足多种需要。但当移动对象数量很大，如一个城市的所有出租车，系统需存储的数据量就会非常庞大。

2）运动轨迹数据类型及存储

移动对象的各个移动位置点从整体上来看就形了一条运动轨迹，因此在技术上可将整条运动轨迹作为存储单元而不仅是只考虑一个一个的位置点数据，这就是移动对象数据的基于运动轨迹数据存储。

在 MOD 中，移动对象的运动轨迹也称为轨迹线，并按照轨迹线表示的不同分为移动函数存储和移动轨迹存储两种情形。

(1) 移动函数存储。通过构造移动对象运动函数表示对象运动轨迹。运动函数采用普通的函数方法，在有效的时间内推算出对象的位置和相应的运动类型。通过运动参数描述了运动特性，如移动对象的开始位置、运动速度及运动方向等。这样，对于运动比较规律的移动对象，如具有固定运行路线的飞机、船舶和列车等，通过存储比较少量的运动参数就能满足存储历史轨迹数据的需要，同时也预测移动对象未来的位置。

(2) 移动轨迹存储。将移动对象各个空间位置点用直线段顺次相连组成一条由折线段组成的运动轨迹。在分析移动轨迹各时间段的情况时可将移动轨迹分段处理。分段的依据可以是移动对象方向的改变或状态的变化，也可以是按照确定的时间间隔进行采样。移动轨迹能够较好地反映移动对象的运动情形，通常可作为移动对象查询等的基本单位，相比于空间位置点数据存储，移动轨迹需要存储的数据量可以大大减少，有利于提高系统性能。

2. 移动对象数据管理环境

移动对象数据产生于移动计算的环境中，与基于固定网络的传统分布计算环境相比，移动计算环境具有其自身的一些特点。

(1) 设备移动性。移动计算环境最突出的特征是相关设备随时间演进不断移动，一个移动设备需要在不同的地理位置点随时联通所需网络，同时在移动过程中也需要保持相应的网络连接。

(2) 频繁间断性。移动设备在移动过程中，由于使用方式、电源供给、无线通信费用和网络条件等因素的限制，一般不能采用持续联网的工作方式，而是主动或被动式的间歇性入网与断接。

(3) 低带宽性。与固定网络相比，无线连接的带宽要小许多。

(4) 网络通信非对称性。由于物理通信媒介的限制，一般无线通信都是非对称的，即在固定服务器结点可拥有强大的发送设备，而移动设备发送能力非常有限，由此导致下行链路(服务器到移动设备)的通信带宽与代价和上行链路(移动设备到服务器)差别较大。

(5) 低可靠性。与固定网络相比，无线通信网络可靠性较低，容易受到各类白、黑干扰而出现各类网络通信故障。

上述情形将会导致基于移动计算环境的移动对象数据库应用系统具有不同于常规数据库应用系统的基本特征。一个简化的 MOD 应用系统架构如图 8-1 所示，它通常包括移动对象客户端和位置管理服务器两个主要部分。

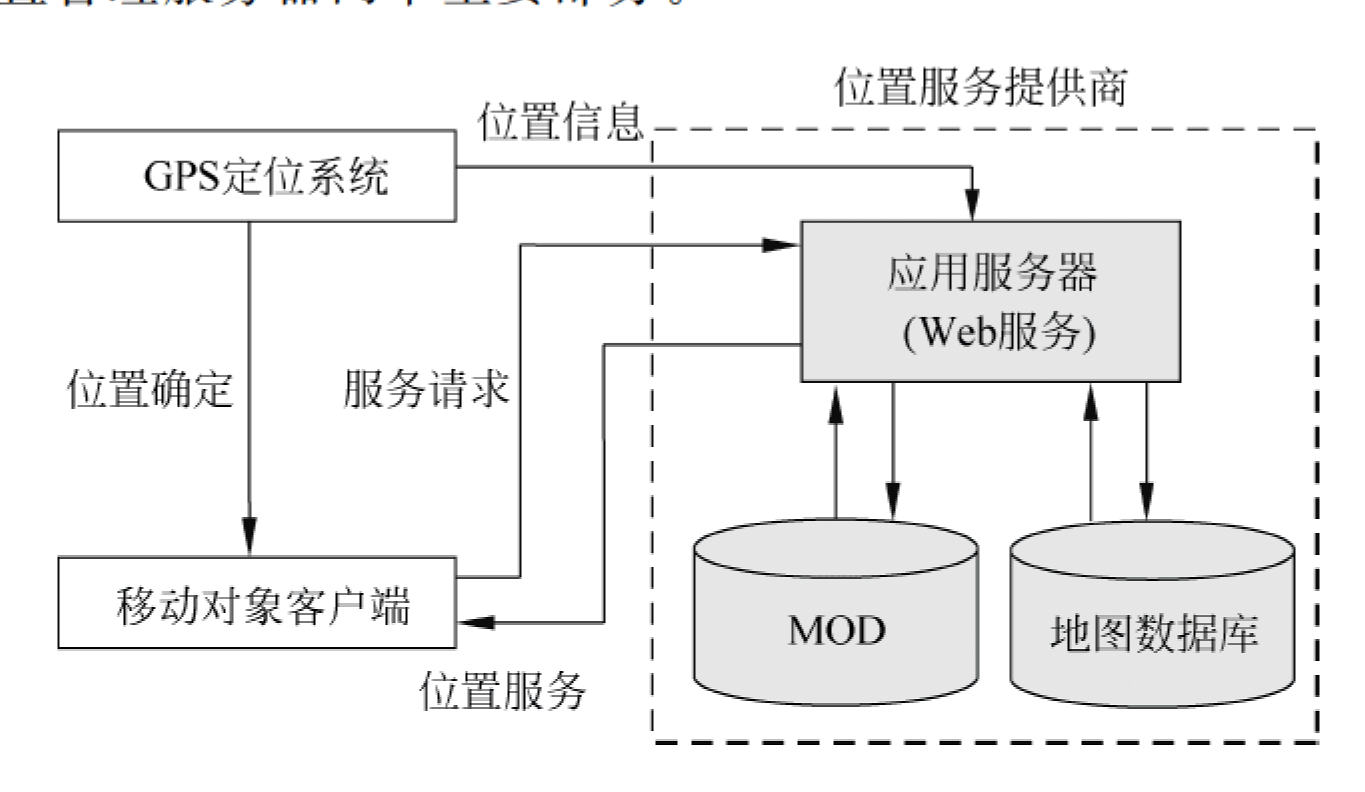


图 8-1　基于移动计算环境的 MOD 应用系统架构

（1）移动对象客户端：例如，使用移动手机通过位置接收装置（GPS接收器）获取自身位置数据，再通过无线通信网络向服务器报告其位置数据。

（2）位置管理服务器：应用服务器接收这些数据并将其存储在MOD中。同时，服务器还将存储其相关数据（如速度或方向信息）以预测移动对象的未来位置。

移动对象客户端可通过无线通信网络向服务器发出位置相关的查询请求，位置应用服务器利用移动对象数据库和地理信息数据库中的信息进行查询处理后再将结果通过无线通信网络返回给移动对象。

3. MOD主要技术领域

目前，移动对象管理主要研究课题和技术领域有下述几个方面。

1）位置建模与数据索引

（1）位置建模。为了对移动对象的位置进行有效的管理，MODS必须能够准确地获取和存储移动对象的当前位置信息，这就涉及移动对象数据的逻辑表示与存储结构，因此需要建立有效的位置数据信息的描述与管理模型。

（2）数据索引。MOD管理着数量非常庞大的移动对象数据，但作为一种新型数据库，目前并没有如同RDB那样的查询优化技术，当然也不能只进行遍历访问查询。如同其他非传统数据库（如XML数据库）那样，为了减小搜索空间和提高查询效率，就只能借助于数据索引技术。移动对象数据索引技术是MOD领域中一个充满挑战性的研究领域，有许多开放性课题尚待进一步深入探讨。

2）数据查询与不确定性处理

（1）数据查询。MOD中的查询目标分为两种，一种是移动对象（如汽车、移动用户等），另一种是静态空间对象（如旅馆、医院等），这两类数据查询需要各自相应的索引结构的支持。MOD数据查询通常都具有位置相关的特性，即查询结果依赖于移动对象所处位置，同一查询请求，如果提交时间及地点不同，返回结果也将不同。典型的查询包括区域查询（查询某个时间段处于某个地理区域的移动对象）、KNN查询（查询离某一点最近的K个移动对象）及连接查询（查询满足条件的移动对象组合）等。

（2）不确定性表示及处理。如前所述，移动对象位置数据管理本质上具有不精确性和不确定性。不管采用哪种位置管理及位置信息更新策略，MOD中保存的位置信息与移动对象实际位置情况都会存在一定偏差。例如，对于周期性位置更新方法，位置数据更新是周期性完成的，在每个更新周期内，数据库中位置数据不变，而实际上移动对象可能已经在此期间离开了原来位置。将位置表示为时间的函数同样也存在位置的不确定性，位置函数仅仅近似地刻画了实际位置变化。此外，系统通过设定阈值的方式来减少位置信息更新代价的策略，实际上也增加了位置的不确定性。如何明确表示这些不确定性以进入计算机进行存储管理，如何根据需要获取数据处理后的不确定性度量以投入实际应用，这些都是MOD技术的基本研究课题之一。

3）位置数据更新

随着无线通信技术与全球定位技术的发展，基于位置的服务能够向用户提供可靠的位置服务信息并得到广泛应用。但在基于位置的应用服务系统中存在移动对象频繁和大量的数据更新，如何有效管理移动对象位置的更新也就成为MOD的重要特征和关键技术。

为有效地减少移动对象位置存储的更新频率，提高通信效率与提供精确的位置服务，在

对移动对象实施更新的过程中，更新开销是衡量移动对象位置更新系统优劣的基本指标，主要包括下述内容。

（1）网络通信开销：由于通信网络带宽限制，移动对象频繁地更新会加重网络开销甚至导致网络通信难以实施。

（2）服务器端开销：传统数据库管理系统难以支持大量频繁更新，如果使用 RDB 等对移动对象位置更新的支持就必须付出巨大额外开销。

（3）客户端开销：由于通信网络带宽的限制，移动对象客户端必须执行复杂计算，但移动设备电量有限，处理能力有限，难以实施有效的数据更新。

讨论移动对象位置数据更新机制需要对移动对象位置数据信息进行形式化描述。

（1）移动对象的形式化描述。可以采用如下方式对移动对象位置信息进行描述和定义。

① 移动对象：设二维空间 $X \times Y$ 位置点 $p=(x,y)$ 集合记为 P，时间域记为 T，则三维时空 $X \times Y \times T$ 中移动对象 MO 表示为三元组 MO$=(p,v,t)$。其中 $p \in P$ 表示移动对象初始位置，$\boldsymbol{v}=(v_x,v_x) \in R^2$ 表示移动对象速度矢量，$t \in T$ 表示 p 与 v 对应时间。

② 移动对象运动方向：运动方向与正北方向即指向地球北极方向的夹角。

③ 服务对象：移动对象服务器发往客户端的服务信息，并记为 SO＝（POLICY，OB，THR）。POLICY∈{'POINT'，'VECTOR'}为更新策略，OB∈{MO}为移动对象位置在服务器上的表示方式，THR∈{THRESHOLD}是移动对象更新阈值。

（2）移动对象位置更新策略。位置数据更新策略以移动对象轨迹建模为基础。

① 基于移动点建模更新：将移动对象的位置建模为固定位置点的时间函数，通常只包含移动对象位置点信息。基于点移动位置更新策略，即点更新策略（POINT POLICY），通常使用常量位置函数预测移动对象在未来某一时刻的位置，如果当前位置值超过预先设定阈值就发起一个更新。点更新策略将移动对象的运动位置信息表示成一系列跳跃的点，是各类更新策略的基础。基于点位置更新策略实现比较简单，适合运动方向难以预知的移动对象，如行人等。

② 基于矢量建模更新：将移动对象的位置建模为随时间变化的线性函数，其中包括移动对象运动的位置、速度与方向等多方面信息。基于矢量（如速度）的移动对象位置更新策略，即矢量更新策略（VECTOR POLICY），通常使用随时间变化的线性函数预测移动对象在未来某一时刻的位置。移动对象从最近更新处做匀速直线运动直到再次发生更新，移动对象的运动表示为跳跃的矢量。基于矢量位置更新策略适合于在确定线路上做近似分段直线运动的移动对象，如汽车、飞机和高铁列车等。

（3）位置更新体系。位置更新体系由移动对象客户端、无线通信网络与移动对象服务器等三部分组成。为描述确定，假定移动对象客户端安装 GPS 定位器，移动对象客户端与移动对象服务器通过无线通信网络相连接，发生断线时，移动对象服务器将采用适当方式解决，同时，移动对象通过自身安装的 GPS 定位器每隔 1 秒报告一次当前实际位置。由于对当前实际位置测定通常都存在一定误差，此时如果移动对象客户端获得的位置数据比服务器端计算出来的更为精确，则客户端就采用由 GPS 定位器获得的精确位置；然后，移动对象服务器等待接收客户端更新请求。客户端向服务器发送更新记录包含当前位置值及其他运动数据，依据这些数据，移动对象服务器就可预测移动对象运动趋势及未来某个时刻位

置。移动对象更新频率取决于对位置信息精度的需求，何时向移动对象服务器发送更新请求通常由客户端确定。移动对象客户端依据发送给服务器的更新数据与服务器发送的位置信息计算预测未来某时刻的位置。客户端将预测的位置值与从 GPS 定位器获得的实际位置相比较，当两者的偏差值大于预先设定阈值时，客户端向服务器发出更新请求，否则，没有移动对象更新发生。移动对象位置数据更新体系如图 8-2 所示。

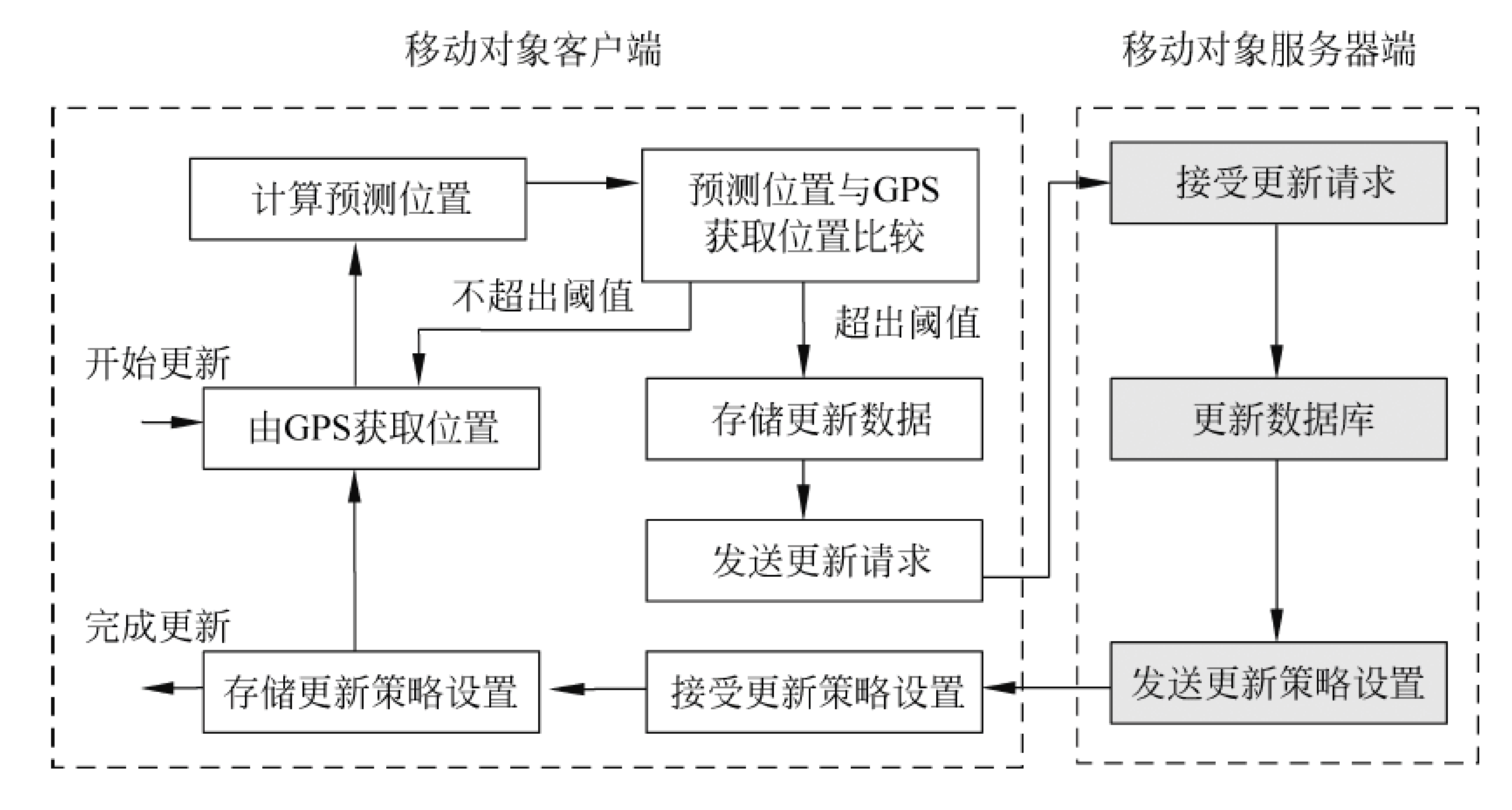


图 8-2　移动对象位置数据更新体系

4. MOD 应用领域

当前 MOD 主要有下述应用领域。

(1) 智能交通系统(交通调度与管理)。交通管理系统可以基于 MOD 管理车辆位置信息，如做出交通堵塞标识以控制交通、预测在未来 10 分钟内通过火车站的车辆数目，出租汽车公司可以根据 5 分钟后车辆可能的位置进行车辆调度等。类似状况还可应用在出租车/警员自动派遣系统、智能社会保障系统及高智能的物流配送系统等。同时，还可应答用户各种位置查询请求，提供交通导航服务，如用户在驾车行驶过程中查询距其最近的加油站和查询到达目的地的最优路径等服务请求。

(2) 位置相关服务(基于位置的广告)。移动运营商位置服务平台可根据移动用户的当前位置发送附近商业企业的广告。例如，当顾客经过某商场范围时，其移动电话中将接收到该商场的促销广告、商品目录和提供电子优惠券等。

(3) 集成旅行服务。旅行社首先通过位置服务平台确定用户所在的位置，其次再根据数据库或互联网提供的信息选出用户所在地的相关信息，提供一整套可定制的、个性化的旅行集成信息，如酒店预订信息、周边餐馆信息、交通信息和文化活动等。

(4) 紧急救援服务。电信服务商基于移动电话网络或 GPS 定位为公众提供紧急救援服务，如美国的应急系统 E911 和欧洲的 E112 服务等。又如，在发生交通或野外事故时，用户可利用移动电话呼叫救援，相关部门就可定位救援者位置并确定最佳救援方式，也可通过移动电话的定位功能为老人和小孩提供跟踪服务。

(5) 数字战场。在战场环境中，作战单元往往需要知道特定区域内移动目标(坦克、直升机、战士等)位置数据或需要对重要移动目标位置进行跟踪监视。战场环境中移动目标位置数据通过无线通信网络、传感器网络或卫星侦察定位等方式向位置服务平台提供。战场

管理系统位置服务平台负责管理战场环境中的各种移动目标并实时回答作战单元的各种查询服务。例如,查询在某地区内有多少友军坦克,查询距离某战士最近的直升机等,回答常规数据库无法回答的查询,如“请告诉距离当前位置最近友军代号及方位”等。

8.2 移动对象数据模型

数据库系统处理能力从逻辑上来讲依赖于相应数据模型的设计与构建。数据模型是数据库的核心元语,它从本质上规定了数据对象的类型、相互关系和数据操作以保持数据库完整性的规则。一个科学严格的数据模型能够定义和控制数据库中将会执行的查询模式和更新操作。同时,基于时间演进动态构造的移动对象信息也需要在数据模型所定义的数据模式中体现出来以利于在 MOD 系统中存储、提取和分析。

8.2.1 移动对象数据建模概述

MOD 产生发展于 20 世纪 90 年代中后期,就当前情形来说,建立一个成熟、高效和统一的移动对象数据模型还具有相当的挑战性。在过去二十多年间,人们根据不同应用背景提出了满足不同实际需求的移动对象数据模型,但这些模型也各有自身的应用边界。

建立移动对象数据模型,通常是先在原理层面研究连续抽象数据模型,再在数据管理层面研究便于描述、存储和处理的离散有限数据模型。

1. 连续模型与离散模型

连续和离散是探讨与时间相关的计算机应用的两个前提。抽象的连续模型是进行逻辑分析的基础,具体的离散是实现计算机处理的前提。两者相辅相成,对于复杂的移动对象数据管理也是如此。

1) 连续抽象数据模型

连续抽象数据模型的基本特征是以位置点的无限集合为初始,着眼于对移动对象自身特征描述表示与相互关系的分析计算。其基本点可以描述如下。

(1) 将 X、Y 平面上移动点的运动描述为由两个空间维和一个时间维组成的三维时空中的一条连续曲线。

(2) 将 X、Y、Z 空间中移动区域运动状态看作是由 3 个空间维和一个时间维组成的四维时空中的一张连续运动曲面。

连续抽象模型实际上就是将所涉及的移动对象运动处理为由一维无限时间域到二维或三维无限空间域的连续映射。例如,对于来袭导弹而言,其飞行轨迹无论能否建立一个有限的表达描述形式,在抽象层面上都需要处理为一条连续的曲线,在相应时间区间内任意时刻总存在一个相应的位置点取值。

连续抽象数据模型实际上可以看作是常规情形下的概念数据模型。作为进行逻辑分析与计算的基础,连续抽象数据模型概念上简洁明确,相关语义易于描述,但不能直接通过计算机系统进行数据的存储和操作。

2) 离散有限数据模型

建立抽象数据模型之后,就需要建立相应的离散有限数据模型以便在相应计算机系统中实现,离散有限数据模型可以看作是连续抽象模型的有限表示。类似于 SDB 中使用多边

形表示区域和使用折线表示河流等,离散有限数据模型的基本点也是“图形近似”。

(1) 通过曲线的直线段近似将二维空间中移动点的运动建模为三维时空中的一个折线段集合(折线)。

(2) 通过曲面的平面块近似将三维空间中移动区域运动建模为四维时空中的一个多面体集合。

相对于连续抽象数据模型,离散有限数据模型的概念更为复杂多样,语义分析处理也更为精细深入。移动对象离散有限数据模型可以看作是常规数据建模中的逻辑数据模型。通过离散模型,就可以为其基础建立起相应的计算机处理意义下的数据结构和数据操作。MOD 中的数据建模通常都是指离散有限数据模型。

如前所述,MOD 属于时空数据库范畴,但其显著特征是,只关注移动对象的位置数据信息,因此可以分别通过基于时空数据和基于位置服务两种方式实现移动对象的逻辑建模。另外,移动对象位置数据与时间演进密切相关。从对时间演进的感知角度考虑,可分为过去、现在和未来 3 种时间类型。当前比较受到认可的逻辑模型多是针对上述 3 种类型中的一种或两种,同时基于 3 种类型的模型也处于开放过程中。

2. 基于位置服务建模和时空数据建模

移动对象在空间的位置随时间的演进而变化,描述和表示移动对象数据频繁更新的特点是移动对象数据建模的基本出发点。如前所述,离散数据模型中的移动点对象运动状态建模为线段集合即折线,这在 MOD 中通常称为轨迹。离散数据模型在计算机中的应用实现方式研究主要是围绕轨迹的频繁更新基本点展开,因此 MOD 中的数据建模在某种意义下可以说实际上就是对移动对象轨迹的建模。在 MOD 轨迹建模领域中存在着两个并行的发展方向,即基于时空数据管理的数据建模与基于位置服务的数据建模。

1) 基于时空数据管理的数据建模

基于时空数据管理的数据建模的基本代表是以德国哈根大学 R. H. Guting 为代表的欧洲项目研究组 Chorochronos 成员开发的时空数据库模型,该模型目标是管理随时间变化的几何形体。早期只针对离散变化情形,其后逐渐地处理连续变化的移动点、线和区域。此模型具有相关的查询语言用于处理历史信息,同时给出了数据类型与相关运算符定义,建立了支持移动点、线和区域的处理算法。该模型特点是相应的数据类型与算法可嵌入到现有的常规数据库模型中,能够有效支持过去数据查询,但对未来查询支持不足。另外,还有以法国国家科研中心 Inria 的研究员 S. Grumbach 为代表提出的时空数据库模型的约束表示模式,该模型主要集中在移动点的处理上,原型系统为 Dedale。

2) 基于位置服务的数据建模

基于位置服务的数据建模的基本代表是以美国 Illinois 大学芝加哥分校的 Wolf Sono 等开发的一个用来存储随时间变化的位置信息模型 MOST (Moving Objects Spatial-Temporal)。MOST 模型着眼于对当前和未来位置信息的预测处理,较好地体现了 MOD 不同于常规数据库的基本特性。由于未来位置数据信息并未产生,因此 MOST 主要是进行预测,它并不直接存储需要频繁更新的位置数据而是存储运动矢量。当移动对象运动矢量(如速度)达到预定的阈值时才会更新数据库所存储的位置数据。MOST 提出动态属性概念和瞬时、连续及持续查询的思想,同时给出了相关的 FTL(Future Temporal Logic)查询语言。

8.2.2 MOST 模型

随着 GPS 技术和无线通信技术的发展，管理移动对象动态位置数据成为可能。快速有效地获取移动对象当前位置数据是进行复杂位置管理的先决条件。移动对象位置数据总是发生变动，为了保证位置数据随时有效，传统方法是周期性刷新数据。但移动对象位置数据频繁变动会给服务器带来繁重的更新开销，同时也会大大增加网络负担，由此形成了研究新的处理方法的实际驱动。当前 MOD 主要采用基于位置函数的建模方式，即将移动对象的位置抽象成时间的函数：Location=$f(t)$，系统根据该函数计算出移动对象在未来任一时刻的位置，移动对象无须周期性地报告当前位置，只在实际位置与计算位置的偏差达到或超过一定阈值时才对数据库进行更新。

基于位置函数建模可以有效降低数据库更新开销，并由此减轻了网络负担。这方面具有代表性的研究成果就是移动对象时空模型 MOST。

MOST 基本思想是引入动态属性概念，将移动对象的位置表示为时间的函数。移动对象的动态属性是基于位置的属性，由 5 个动态子属性构成：L. route、L. X. updatevalue、L. Y. updatevalue、L. updatevalue、L. speed。

MOST 模型也存在着一定的局限性。由于所采用的简单函数表达能力有限，动态属性只能表达移动对象在未来较短时间内的移动轨迹，而对较长时间移动对象轨迹的表示就力不从心，同时，MOST 不支持历史位置数据查询。

1. MOST 中全轨迹

如前所述，移动对象轨迹通常都使用一系列首尾相连的线段即折线来表示，折线中每个线段的位置通过线性插值技术获得，每个线段还使用一个阈值表示轨迹折线上位置与移动对象实际位置偏差。移动对象轨迹的一般形式化定义如下。

三维时空 $X \times Y \times T$ 中给定移动对象的轨迹 Tr 是由点序列(x_1, y_1, t_1)，(x_2, y_2, t_2)，…，(x_n, y_n, t_n) $(t_1 < t_2 < \cdots < t_n)$构成的一条折线，折线中每一段都为直线段。

轨迹 Tr 在二维空间 $X \times Y$ 平面上的投影定义为路径 Rr，如图 8-3 所示。

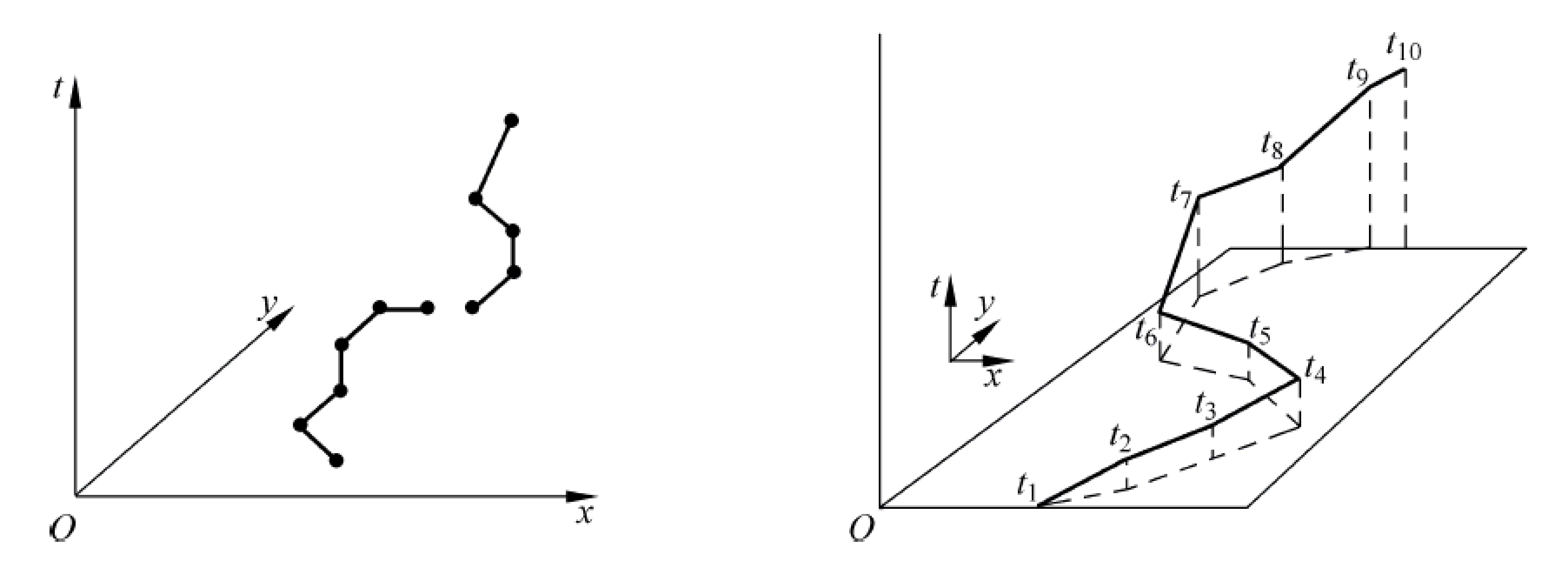


图 8-3 移动对象的轨迹表示

移动对象位置信息可以通过轨迹概念将其表示为一个关于时间的(隐式)函数，其语义为当移动对象在时刻 t_i 位置坐标为(x_i, y_i)时，在时间区间$[t_i, t_i+1]$上由点(x_i, y_i)到点(x_{i+1}, y_{i+1})且以速度 v_i 做匀速直线运动。

速度 v_i 可以通过下述公式计算获得：

$$\vec{v}_i = \frac{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{t_{i+1} - t_i}$$

在时间区间$[t_i, t_{i+1}](1 \leqslant i < n)$内，移动对象在轨迹 Tr 上两点(x_i, y_i)与(x_{i+1}, y_{i+1})之间的位置通过线性插值方法获得。

在 MOST 中同时考虑未来时间轨迹和过去时间轨迹，因此需要将上述一般形式的轨迹扩展为下述的包含移动对象未来与过去轨迹的全轨迹。

1）基于 MOST 模型的未来轨迹

设三维时空 $X \times Y \times T$，移动对象未来轨迹 Tr 是一个由下述元素组成的七元组：

$$Tr = (\text{Moid}, \text{Futnumber}, \text{Loc}_0, \text{t}_0, \text{Direction}, v_0, \text{Uncertainty})$$

（1）Moid：移动对象标识符。

（2）Futnumber：未来轨迹段标识符。

（3）Loc_0：更新初始位置。

（4）t_0：更新初始时间。

（5）Direction：轨迹方向。

（6）v_0：更新初始速度，由后述速度预测方法求出。

（7）Uncertainty：更新阈值。

为得到移动对象的全轨迹需要在移动对象位置发生更新的时刻，将更新以前的位置信息存储到历史数据库中再做更新。历史数据库是存储过去轨迹的集合。

2）基于 MOST 模型的过去轨迹

设三维时空 $X \times Y \times T$，移动对象过去轨迹 Tr 是一条由下述六元组构成的折线：

$$Tr = (\text{Moid}, \text{Pasnumber}, \text{StartLoc}, \text{StartT}, \text{EndLoc}, \text{EndT})$$

（1）Moid：移动对象标识符。

（2）Pasnumber：过去轨迹段标识符。

（3）StartLoc：移动对象开始位置。

（4）StartT：移动对象开始时刻。

（5）EndLoc：移动对象结束位置。

（6）EndT：移动对象结束时刻。

邻近的两个点由直线段相连接。轨迹 Tr 在二维 $X \times Y$ 平面上的投影称为路径 Rr。两个六元序列之间任何时刻的位置可以通过线性插值得到。

由此可知，基于 MOST 模型扩展的移动对象全轨迹就是在记录当前与未来位置信息时，同时存储更新以前的位置数据，由此不仅实现了移动对象过去、现在和未来移动对象全轨迹建模，而且在一定程度上还解决了轨迹建模不确定性的问题。

2. 轨迹预测方案

在以下讨论中，设移动对象 M_0 在未来时刻 T 的预测位置为 PredLoc，实际位置为 LacLoc，更新时刻为 t_0，移动对象的位置为 Loc_0，预测初始速度为 v_0，在离上次更新的时间间隔为 $t' = t + t_0$ 时，移动对象 M_0 的预测位置为：$\text{PredLoc} = \text{Loc} + v_0 t$。此时，为得到移动对象未来轨迹上的预测位置 PredLoc，关键之处是需要对初始速度 v_0 的预测。

移动对象轨迹预测技术通常采用两种速度预测模型，即延时模型与滑动平均模型。

将 t_{n+1} 设为未来采样时刻，在 t_{n+1} 时刻到来之前，需要对速度 v_0 进行预测。由于 v_n 为

矢量,其预测包括标量值和方向。

(1) 延时模型

$$\hat{v} = v_{n-1}$$

式中,$\hat{v}$为未来时刻速度;v_{n-1}为最后一个历史速度。

未来时刻速度标量值和方向都与最后一个历史速度 v_{n-1} 的标量值和方向一致。

(2) 滑动平均模型

$$\hat{v} = \frac{1}{P}\sum_{i=1}^{p} v_{n-1}$$

式中,$\hat{v}$为未来时刻速度;P 为历史速度的个数。

未来时刻速度数量值是前 P 个历史速度的数量的平均值,其方向与最后一个历史速度 v_{n-1} 的方向一致。实践表明,延时模型对移动对象运动状态改变适应更快,而滑动平均模型对速度标量值的预测比较准确。

采用上述两种基于历史速度预测未来速度的方法,虽在一定程度上对速度做了预测,但预测精度值难以满足应用需求。为了对未来速度做出更为准确的预测,还需要建立起带有滑动因子 α 的速度预测模型。

(3) 滑动因子预测模型。此模型也采用历史的速度去预测未来速度,基本点在于滑动因子 α 的选择,滑动因子 α 决定了速度的预测。α 通过以下公式计算:

$$\begin{cases} \alpha = \dfrac{\alpha_1 + \alpha_2}{2}, & 0 \leqslant \alpha \leqslant 1 \\ \alpha = 1, & \alpha > 1 \\ \alpha = 0, & \alpha > 0 \end{cases}$$

$$\alpha_1 = \frac{\sum_{t=3}^{n}(v_t - v_{t-2})(v_{t-1} - v_{t-2})}{2\sum_{t=3}^{n}(v_{t-1} - v_{t-2})^2}$$

$$\alpha_2 = \frac{3}{4}\left[\frac{\sum_{t=3}^{n}(v_t - v_{t-1})(v_{t-1} - v_{t-2})}{\sum_{t=3}^{n}(v_{t-1} - v_{t-2})^2 + 2\sum_{t=4}^{n}(v_t - v_{t-1})(v_{t-2} - v_{t-3})} + 1\right]$$

式中,t 为当前时刻;v_t 为当前时刻 t 的速度,$v_{t-i}(i=1,2,3)$为当前时刻 t 之前的第 i 个时刻速度;n 为计算滑动因子 α 的时刻总数。设采样时刻移动对象各个历史位置如下。

$$(x_{t-i}, y_{t-i})、\cdots、(x_{t-4}, y_{t-4})、(x_{t-3}, y_{t-3})、(x_{t-2}, y_{t-2})、(x_{t-1}, y_{t-1})、(x_t, y_t)$$

则历史速度计算可由前述定义中的公式得出:

$$v_i = \frac{\sqrt{(x_{t-i+1} - x_{t-i})^2 + (y_{t-i+1} - y_{t-i})^2}}{\Delta t}$$

式中,Δt 为两个连续采样点的时间间隔且 $0 \leqslant i < t \leqslant n$, $n(0 \leqslant n < t)$为历史记录的长度。

设 $N=(2-\alpha)/\alpha$,则预测未来时刻 $t+1$ 的速度速度$\hat{v}_{t+1}$可由下述公式计算:

$$\hat{v}_{t+1} = \alpha v_i + \alpha(1-\alpha)v_{i-1} + \alpha(1-\alpha)^2 v_{i-2} + \cdots + \alpha(1-\alpha)^N v_{i-N}$$

如果要预测未来时刻 $i(i>t+1)$的速度,首先计算$\hat{v}_{t+2} = \alpha v_{i+1} + \alpha(1-\alpha)\hat{v}_{i+1}$,其次用$\hat{v}_{t+2}$去

估算$\hat{v}_{t+3}$，$\hat{v}_{t+3}=\alpha v_{i+2}+\alpha(1-\alpha)\hat{v}_{i+2}$，再用$\hat{v}_{t+3}$估算$\hat{v}_{t+4}$的值，$\hat{v}_{t+4}=\alpha v_{i+3}+\alpha(1-\alpha)\hat{v}_{i+3}$……由此就可以得到未来时刻 i 的速度$\hat{v}_t$。

MOST 将当前速度处理为未来时刻预测的速度，这在轨迹预测过程中存在着相当的不精确性。为提高预测质量，人们还对滑动平均模型与滑动因子速度预测模型进行了多种改进。

3. 轨迹更新策略

为解决移动对象全轨迹建模中的不确定性与不精确性，移动对象轨迹更新策略需要选择合适的更新方式，在降低系统资源占用率的同时保证查询结果具有一定的准确性。为此，在 MOST 建模基础上研究者们提出了两种移动对象轨迹更新策略。

1）速度更新预测策略

在行程开始时，移动对象 MO 发送给 MOD 管理系统一个不确定的阈值，并将其存储在 Uncertainty 中，在整个行程中保持不变。一旦移动对象 MO 的偏差超过 Uncertainty 就会引发更新。每次更新只需要包含移动对象当前的位置、速度和方向。由于此时阈值是随机给定的，难以保证查询结果得到优化。

2）自适应预测策略

在行程开始时，移动对象 MO 发送给 MOD 管理系统一个随意选择的初始化阈值 TH_1。当偏差达到 TH_1 时，移动对象 MO 就发送更新给数据库。每次更新都包含有移动对象的当前位置、速度、方向和一个新的阈值 TH_2。数据库把这个新阈值放进 Uncertainty 子属性中。TH_2 可以通过下述方式获得。

设 t_1 表示从行程的开始到移动对象的偏差第一次达到 TH_1 所经过时间元数，i_1 表示在同样时间间隔里的偏差代价，令 $a_1=\frac{2i_1}{t_1^2}$，则 $TH_2=\sqrt{\frac{2a_1C_1}{1+2C_2}}$ 其中，C_1 是更新代价；C_2 是单位不确定代价。当偏差到达 TH_2 时也会发送一个类似的更新，所不同的是 TH_3 取为

$$TH_3=\sqrt{\frac{2a_1C_1}{1+2C_2}},\quad 其中\ a_2=\frac{2i_2}{t_2^2}$$

式中，i_2 是从第一次更新到第二次更新之间的偏差代价；t_2 表示从第一次位置更新以来经过的时间单元数。

由于 a_2 可能与 a_1 不同，TH_3 也可能与 TH_2 不同。当移动对象的偏差达到 TH_3 后就应当发送另一个包含 TH_4 的更新，TH_4 的计算方法与前面相同。

由此可见，自适应预测策略在移动对象的每次更新时都会提供一个新的阈值。这是一个优化阈值，通过基于信息代价方法得到，从而使得直至下次更新之前单位时间内的总信息代价最小。移动对象需要估计下次更新的产生时刻，也就是什么时候偏差达到阈值。而未来的偏差在进行下次更新时刻估计时是未知的，需要自适应预测策略根据过去的偏差来预测未来的偏差。由于对偏差的预测是不相同的，因此阈值每次更新时计算得出的值也是不同的，从而保证了预测的精确度。

8.3 移动对象数据查询

如前所述，移动对象数据就是相应移动对象随时间演进而改变所处位置的数据，因此时间是描述和查询移动对象数据的入口和标志。通常可将时间分为“过去”“现在”和“未来”

3 种类型，由此在 MOD 中数据查询也可以此分为基于“过去”“现在”和“未来”时间的 3 种基本类型。从时间的计算机描述考虑，主要的时间数据类型是时间点和时间段(时间区间或时间期间)。移动对象数据查询就是在给定时间后，查询满足相应空间关系的移动对象。移动对象基本数据类型是空间位置(坐标)和轨迹，移动对象相互间的空间关系主要有度量关系、序关系和拓扑空间关系，其中拓扑空间关系是空间关系讨论中的重点和难点。将上述移动对象的时间和空间特性描述整合起来，就使得移动对象数据呈现出复杂多样的查询情形。

8.3.1 基于时间点查询

基本的时间数据类型是时间点(时刻)。基于时间点的查询也称为瞬时查询，其特征是查询在某一特定时间点时位于给定空间范围内的所有移动对象。在移动对象抽象数据模型中，当给定的二维空间区域随时间变化时就构成三维时空中的一个曲面体，给定时间点 t_0，就相当于给出了该曲面体的一个剖切面，从直观上来看，就相当于打开了曲面体的一个窗口，因此基于时间点的查询更多时候也称为“窗口”查询。

1. 时间点查询

设 t_0 和 R_0 分别是给定的时间点和空间范围，下面给出时间点查询的形式化定义。

二元组 $Qw=(t_0,R_0)$ 表示一个基于时间点的(窗口)查询，其语义为查询在时间点(时刻) t_0 位于空间窗口 R_0 的所有移动对象，其中，t_0 可以是过去、现在或未来的时刻点。

根据时间点 t_0 表示过去、现在和未来的不同时间语义，时间点查询有下述 3 种情形。

(1) 过去时间点窗口查询：查询历史数据，对于存储在数据库中的所有数据都可以通过遍历或索引实现相应查询过程。

(2) 现在时间点窗口查询：通过无线定位系统直接判定移动对象是否位于给定窗口之内。

(3) 未来时间点窗口查询：需要根据已有的位置函数计算移动对象在未来某个时间点是否位于给定的窗口之内，这是一个对未来位置进行预测的过程，体现了移动对象数据管理过程不确定性的基本特征。

2. 时间点查询类型

按照空间数据类型和空间关系而言，基于时间点的移动对象查询有不同的查询类型。

1) 基于坐标查询

基于坐标查询时，空间查询结果是相应移动对象的位置坐标或是满足给定位置坐标要求的移动对象。

(1) 点查询：如查询特定飞机、船舶或汽车在给定 t_0 时刻的位置。

(2) 距离查询：如查询距离特定移动对象在当前位置 10km 之内的加油站。

(3) 最近邻查询：如查询距离特定移动对象当前位置最近的高速公路服务区。

最近邻查询是 MOD 中最具特色的典型查询之一，其一般形式为 k 最近邻查询(KNN)即查询 k 个距离给定移动对象最近的空间对象，而上述可以看作是 $k=1$ 的特殊情形。

2) 基于轨迹查询

查询结果是与给定轨迹线具有相应关系的移动对象。

(1) 拓扑查询：如查询在 2017 年 3 月 29 日凌晨 5 时经过(Cross)中山五路的出租汽车。

(2) 导航查询：如查询 C3051 次航班飞行在 2017 年 9 月 21 日 13 时的飞行速度。这

里，飞机速度和方向等动态信息并不显式地存储在数据库中，但可由相应轨迹线推导得出。

8.3.2 基于时间段查询

基于时间段的查询就是对于给定的时间段，查询位于给定空间范围内的所有移动对象。按照给定空间范围 R_0 是否在时间段$[t_1,t_2]$内发生改变可分为下述不同的情形。

1. 移动对象范围查询

基于时间点窗口查询可推广为基于时间段的窗口查询。例如，查询在 2017 年 5 月 8 日上午 8 时至 10 时通过广州市科韵路收费站的所有移动对象(汽车)。

设$[t_1,t_2]$和 R_0 分别为给定的时间段和空间范围，下面给出范围查询的形式化定义。

二元组 Qw=($[t_1,t_2]$,R_0)定义了一个移动对象范围查询，其语义为在$[t_1,t_2]$内通过窗口 R_0 的所有移动对象，而$[t_1,t_2]$可以是表示过去或未来的时间。

移动对象范围查询可看作是时刻点不断变化的基于时间点的窗口查询，其特征是时间点 t 在给定时间段$[t_1,t_2]$内变化，而空间窗口 R_0 不变。

由于时间段用于表示过去时间和未来时间，移动对象范围查询可以有下述两种实现方式。

(1) 过去时间段范围查询。使用数据库中存储的历史数据信息，查询每个移动对象当时间点 t 在过去时间段$[t_1,t_2]$内变化时的移动轨迹，所有轨迹线与查询窗口 R_0 相交部分上的移动对象集合就是查询结果。

(2) 未来时间段范围查询。使用已有移动函数计算出每个移动对象在未来时间段$[t_1,t_2]$变动时的移动轨迹，所有轨迹线与查询窗口 R_0 相交部分上的移动对象集合就是查询结果。

2. 移动对象连续查询

在实际应用中，移动对象范围查询中的空间窗口也随时间点 t 在$[t_1,t_2]$内的变化而发生改变。例如，在数字战场中需要查询在发射后 10～40s 内距离导弹 5km 范围内的敌方反导雷达站；在当前时间 30min 之内距离一辆运行车辆在其 3km 范围内的加油站等，这些都可以看作是引入移动对象连续查询的应用实例背景。

设$[t_1,t_2]$和 R_1,R_2 分别是给定的时间段和空间范围，下面给出连续查询的形式化定义。

二元组 Qcw=($[t_1,t_2]$,$[R_1,R_2]$)定义了一个移动对象连续查询，其语义为时间点 t 在$[t_1,t_2]$内变动时，通过窗口 R 在$[R_1,R_2]$变动时所经历的窗口的所有移动对象，其中，时间段通常是基于查询工作点之后的未来一段时间。

移动对象连续查询实际上描述了一种对于实际查询范围的不确定性预测，因此属于未来时间查询范畴。

移动对象连续查询的基本特点是，查询窗口 R 随着时间点 t 的变化而改变，实际上相当于基于时间点窗口查询结果的一个“连续”变动序列，查询结果当然就是这个序列中所有结果的总和。基于这样的考虑，在移动对象连续查询的具体实现过程通常是将其离散化为多个由固定的更小时间段(将其“看作”时间点)上的时间点查询。

8.3.3 最近邻查询

实际应用中有两类重要的查询类型。其一，如一个在街道上的确定行人，需要查询距离自己最近的出租车；其二，如一辆在街道上的确定出租车，需要查询对于哪些行人来说自己

是距离他们最近的出租车。前者称为“最近邻查询”，而后者则就是“逆最近邻查询”。

设 q 是一个给定的参考点，S 是被查询对象集合，$D(q.p)$ 表示 q 与 p 的距离函数，下面给出最近邻查询的形式化定义。

基于 q 最近邻查询结果是在给定时间点或时间段内 S 中的如下子集 $\mathrm{Qnn}(q)$：

$$\mathrm{Qnn}(q) = \{s \mid s \in S \wedge (\forall p \in S, \quad D(q,s) \leqslant D(q,p))\}$$

由此可知，最近邻查询基本特征就是给定一个参照点，查询在某一时刻或时间期间内存在哪些对象与该参照点距离最近。

如前所述，最邻近查询中通常的类型是 K 邻近查询（KNN），即查找最靠近查询点的 K 个对象。而上述的逆最近邻查询（RNN）从逻辑上可以看作最近邻查询的“逆”描述，其主要查询的最近邻是参照点（查询点）的移动对象，对于查询结果中的每一个点，距它最近的点就是查询点。

显然，最近邻查询的技术实现与查询对象集合 S 和距离函数 D 有关。在实际查询过程中通常以 R-tree 树索引为基础，采用的空间对象集合是 MBR 集合，采用的距离函数是欧几里得距离函数。

最近邻查询在逻辑上可以看作是属于时间点查询范畴，但由于其理论原理和技术实现的特定性，在移动对象数据查询过程中通常单独作为一种查询类型进行探讨。

8.4 移动对象数据索引

MOD 管理着众多的移动对象，而每个移动对象又对应着众多的移动位置或移动轨迹，同时由于还管理着历史数据，因此 MOD 中通常存储着数量庞大的移动对象数据信息。在查询处理时如果只依靠遍历扫描，必然会影响和减低系统性能。提高数据库查询性能的基本途径就是对其管理的数据建立相应的数据索引。从发展应用实践来看，移动对象数据索引技术一直都是 MOD 技术研究与开发的重要领域之一。迄今为止，人们从基本的空间索引和时态索引出发，提出了众多的移动对象数据索引方法。主要情形如图 8-4 所示。

移动对象数据索引技术可按照不同角度进行分类。

(1) 按照查询过程中的时间数据类型：分为基于时间点和基于时间段的数据索引。

(2) 按照用户对查询的时间整体界定：分为基于过去、当前和未来时间的数据索引。

(3) 按照移动对象涉及空间的移动限制：分为基于位置查询和基于轨迹线查询的数据索引。

如前所述，MOD 最突出的特征是“频繁更新性”，因此，无论从什么角度和按照什么关系构建移动对象数据索引都需要尽可能地体现出这种特性，避免由于“少量”数据的频繁更新导致整个数据索引结构的经常性变动，即所构建索引结构应当具有“增量式”更新的品格。

移动对象数据是一种特殊的时空数据，时空数据主体是其中的空间数据信息。空间数据查询有成熟的 R-tree 索引技术，特别是 R-tree 在更新过程中保持“平衡”的增量式更新方法更具有独到的研究与应用价值。基于这样的考量，现有移动对象索引方法大多都是借鉴于空间数据索引特别是 R-tree 技术发展而来的。作为一种经典的空间索引技术，R-tree 自 1984 年由 Guttman 提出后，经过数十年发展，不断产生和拓展的各类变体已经形成了一个

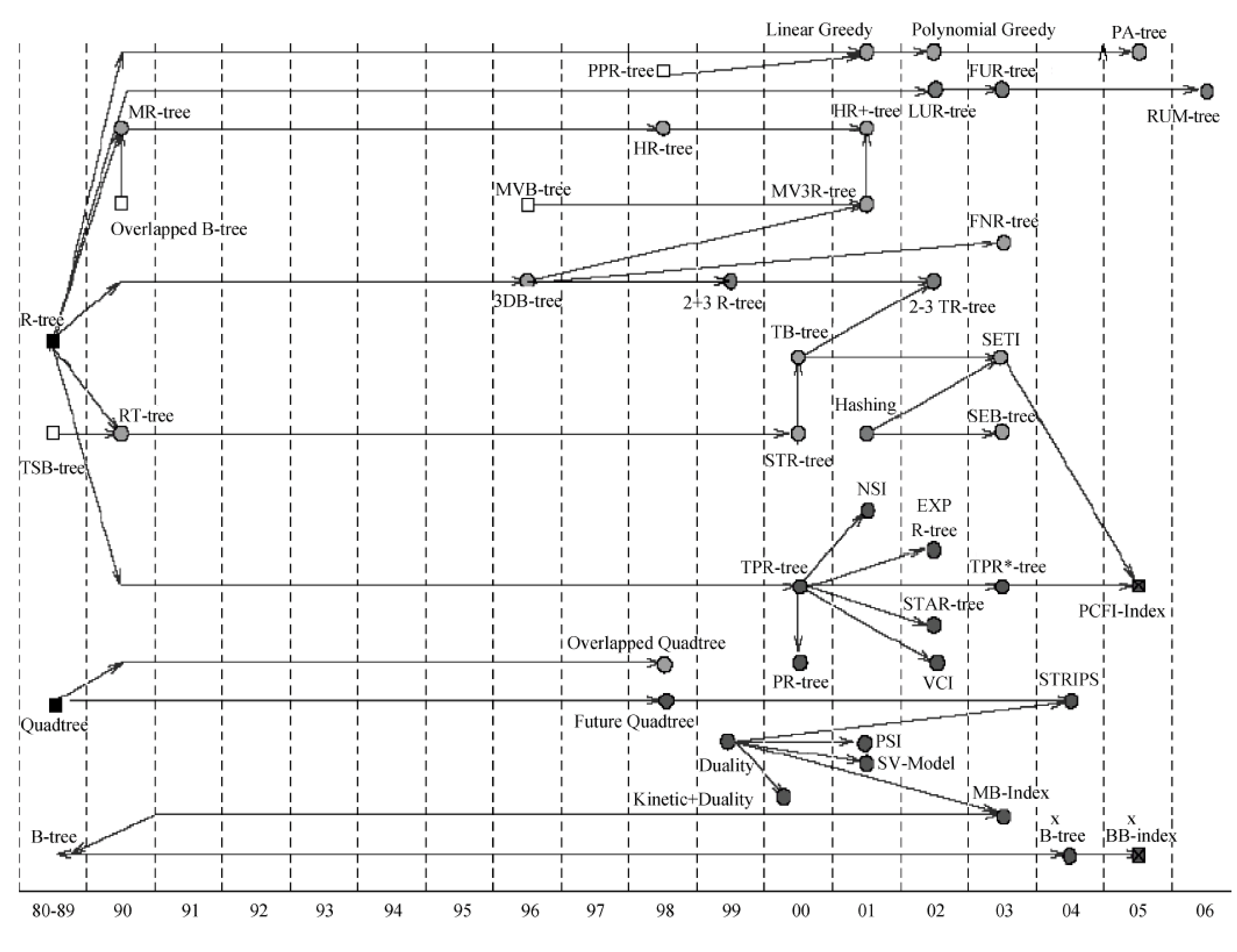


图 8-4　移动对象索引技术

枝繁叶茂的 R-tree 家族。多维空间索引 R-tree 如同一维线性索引 B-tree 树一样，从其覆盖的广度和深度来看都是其他任何索引技术难以比肩的。

用户查询需要使用数据索引，而移动对象历史数据与当前及未来数据的存储管理与检索通常具有较大差异。用户提供的查询时间首先需要厘清是属于“过去”“当前”或“未来”中的哪种情形，因此将移动对象数据索引按照“过去”“当前”或“未来”进行划分就成为基本的分类考虑。现有索引结构通常只支持其中一种或两种类型查询，研究在一个索引结构上同时支持过去、当前和未来的查询仍然是本领域中的开放课题之一。

8.4.1　当前和未来时间索引

查询当前数据及对未来数据进行适当预测是 MOD 中数据查询的重要特色。这类查询索引可采用基于参数化空间的方法进行构建，主要思想是使用时间 MBR 函数描述相应的移动对象在时空中的变化，将具有时空特征的移动对象归入到同一限定矩形中。目前已提出多种属于该类型的移动对象数据索引，这里简要介绍其中的 TPR-tree。

作为基于 R^*-tree 的索引技术，TPR-tree 具有下述特点。

(1) 能够有效索引在 $d(\leqslant 3)$维空间中移动点对象的当前位置。

(2) 能够通过考虑时空对象的速度与方向预测时空对象在较近未来的大致位置。

(3) 通过考虑在索引树结构中可计算的位置以减少时间函数的频繁更新。

(4) 通过更新算法自动调整以适应于一个动态变化数据集。

1. TPR-tree 基本思想

TPR-tree（Time-Parameterized R-tree）的基本点是在 R*-tree 基础上引入时间参数 TMBR（Time-Parameterized MBR）的基本概念。

设移动对象在时刻 t 的位置函数 $x_i(t)=x_{iref}+v_i(t-t_{ref})$，其中 $x_i(i=1,2)$ 分别表示二维欧式空间中点 P 的横坐标和纵坐标，v_i 为移动对象在 x_i 方向上的移动速度。通过位置函数对移动对象位置进行建模，可以实现通过计算完成对当前位置查询和对未来位置预测，减少索引结构的频繁更新。同时，参考位置（$x_{i\,ref}$）和速度向量（v_i）不仅可用于记录和描述对象的运动轨迹，还可以时间函数的形式在索引中表示 MBR 的各个坐标数值。

下面通过实例说明 TPMBR 概念和 TPR-tree 的构建。

(1) 设有如图 8-5(a)所示的 7 个移动对象，图中箭头表示移动方向，为了简单，每个箭头后的数字既表示该移动对象的编号也表示其移动速度。

(2) 按照常规 MBR 构建技术得到给定 7 个移动对象的 3 个 MBR 如图 8-5(b)所示。这里，每个移动对象实际上都带有各自的移动方向和移动速率。此时设时间点 $t=t_0$ 时状态为初始状态。

(3) 假设在时间点 $t=t_1$ 时，按照原定位置移动函数，相应移动对象移动到了如图 8-5(c)所示的位置，原先 MBR 也变化为图 8-5(c)中所示相应情形。

(4) 按照 R*-tree 的思想，此时选取如图 8-5(d)所示的 MBR 更为合适。

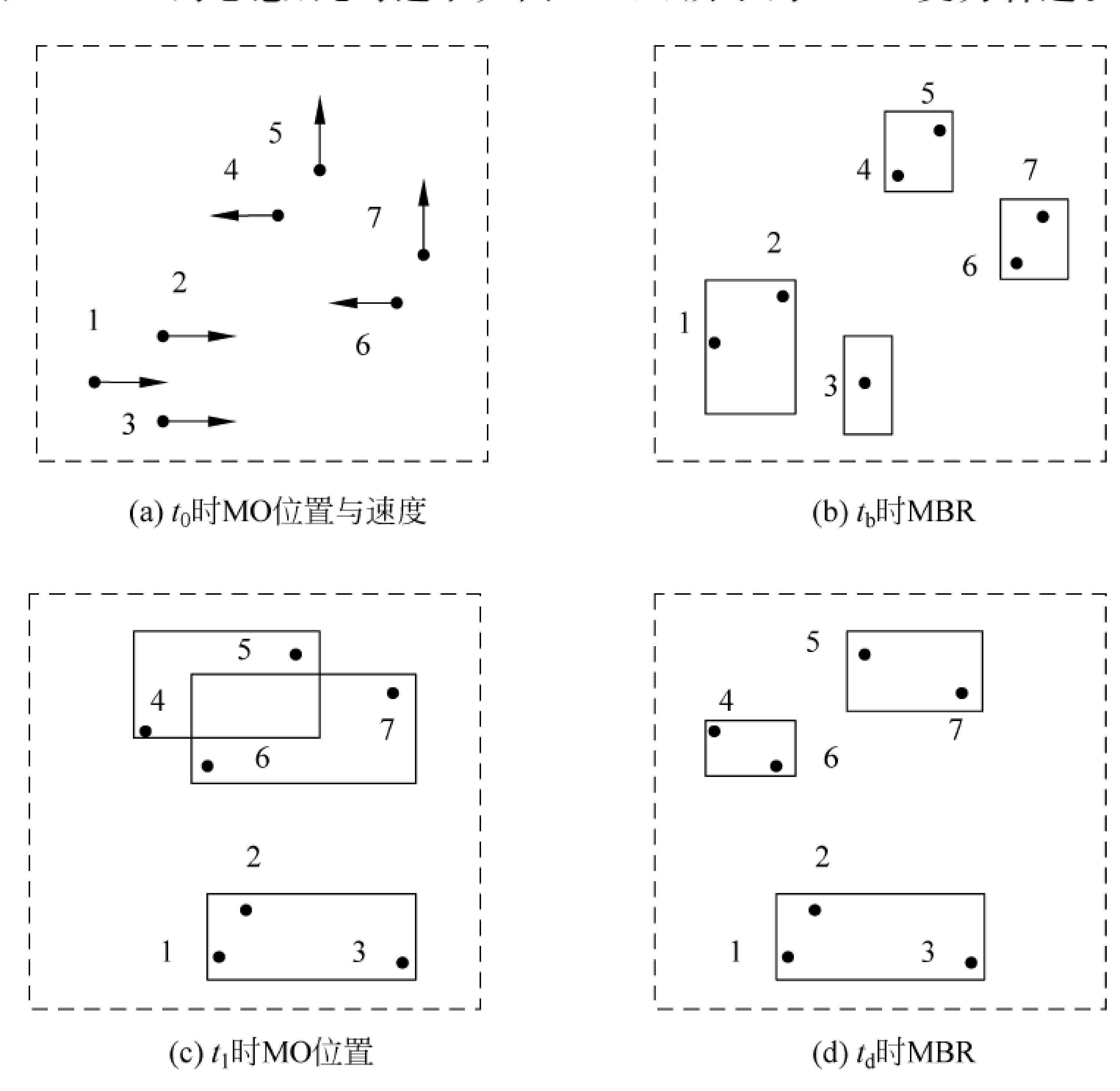


图 8-5　TPMBR 基本结构

上述分析实际上就是将常规 MBR 所包含移动对象中最大的 v_i 分配给 MBR 对应的“上边”和“右边”，最小的 v_i 配置到 MBR 对应的“下边”和“左边”，此时每个 MBR 的 4 条边就分别对应一个速度参数，而这 4 个参数又可以组成一个边分别平行坐标轴的“矩形”，这

样，每个 MBR 都配置了一个速度参数矩形 VBR。从底层上考虑，每个移动对象对应一个 MBR(移动点对象的 MBR 退缩为一个点)，因此可以认为，在 TPR-tree 中每个移动对象实际上都是其形状 MBR 和速度 VBR 的二元组。由此可得到 TPR-tree 中的移动对象如下的形式化建模。

移动对象建模：移动对象建模为二元组 $O=(O_r,O_v)=(\mathrm{MBR},\mathrm{VBR})$，其中

(1) O_r：O 的最小限定矩形，即常规 MBR。

(2) Ov＝VBR＝$(Q_{v_1-},Q_{v_1+};\ Q_{v_2-},Q_{v_2+})$：$Q_{v_1-}$、$Q_{v_1+}$ 分别表示 MBR 的"左边"和"右边"移动速度，Q_{v_2-}、Q_{v_2+} 分别表示 MBR 的"下边"和"上边"移动速度，"＋"和"－"分别表示速度方向与坐标轴相同或相反。

在如图 8-6 所示情况下，4 个移动对象 a、b、c、d 对应 VBR 分别如下：

a：VBR＝(1,1；1,1)　　　　b：VBR＝(－2,－2；－2,－2)

c：VBR＝(－2,0；0,2)　　　d：VBR＝(－1,－1；1,1)

而两个目录 MBR 对应的 VBR 分别为：

N_1：VBR＝(－2,1；－2,1)　N_2：VBR＝(－2,0；－1,2)

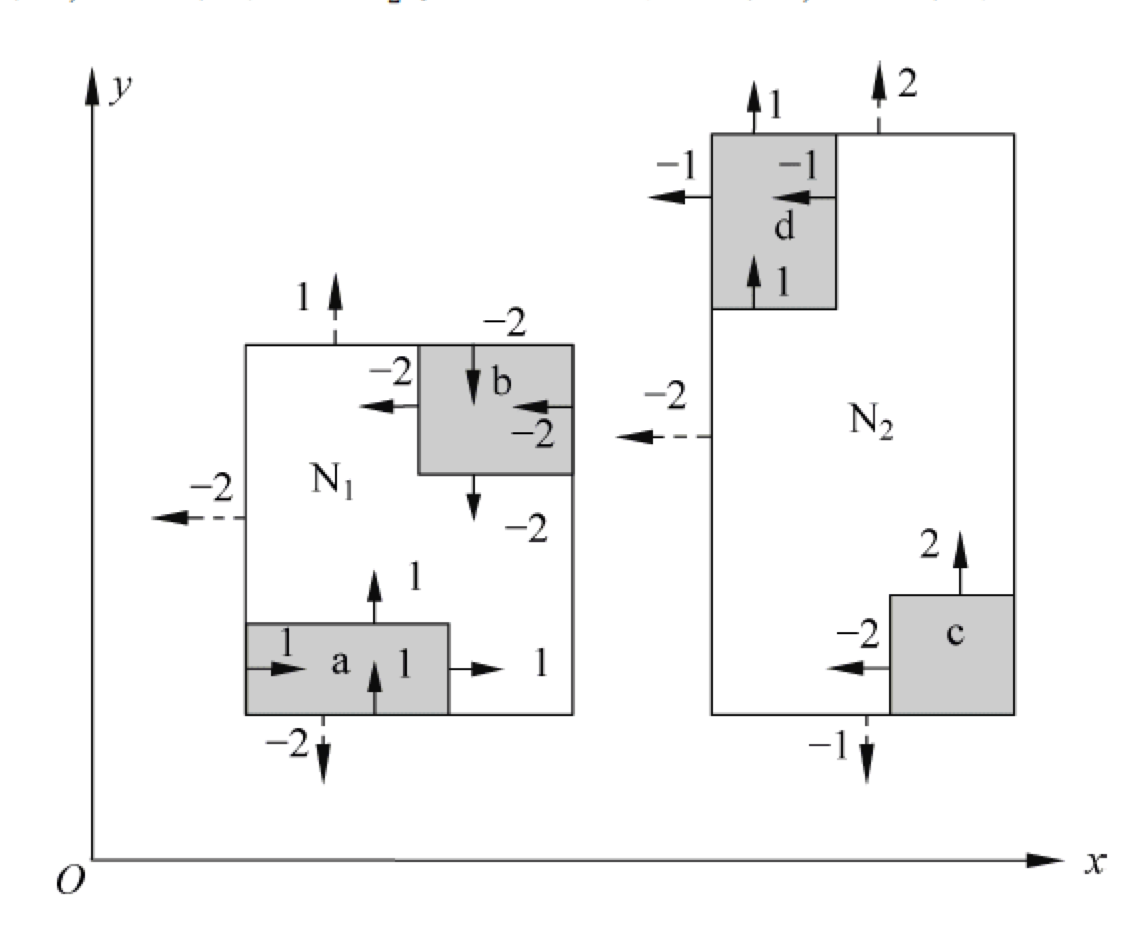


图 8-6　MBR 和对应的 VBR

在构建 TPR-tree 时，可以建立一个"保守边界矩形框"覆盖一个移动对象集合。保守矩形框的下界根据所覆盖对象的最小速度设置，而上界根据所覆盖对象的最大速度设置。因此，保守边界框不会收缩，保证所覆盖的移动对象。为了避免边界矩形框变得很大，无论在什么时候更新对象位置，都需要重新计算所在路径上的所有边界框。

2. 查询类型

移动对象的二维空间 MBR 表示为 $R=([a_{1\min},a_{1\max}],[a_{2\min},a_{2\max}])$，其中 $a_{i\min},a_{i\max}$ 是 R 在第 i 维投影的最小值和最大值，$a_{i\min}<a_{i\max}(i=1,2)$。设 R、R_1、R_2 分别为二维空间中的 MBR，t、$t_{\min}$、$t_{\max}$ 为不小于当前时刻的时间点。根据给定空间范围，TPR-tree 支持下述 3 种查询类型。

(1) 时间片查询：$Q=(R,\ t)$，查询时刻 t 时位于 R 内的所有移动对象。

(2) 窗口查询：$Q=(R,\ t_{\min},\ t_{\max})$，查询时刻段$[t_{\min},\ t_{\max}]$内位于 R 内的所有移动对象。

(3) 移动查询：$Q=(R_1,R_2,t_{\min},t_{\max})$，查询运动轨迹包含在连接 $t_{\min}$时刻的 R_1 与 $t_{\max}$时

刻的 R_2 的不规则四边形内的所有移动对象。

一维空间 x 中 TPR-tree 三类查询示例如图 8-7 所示。

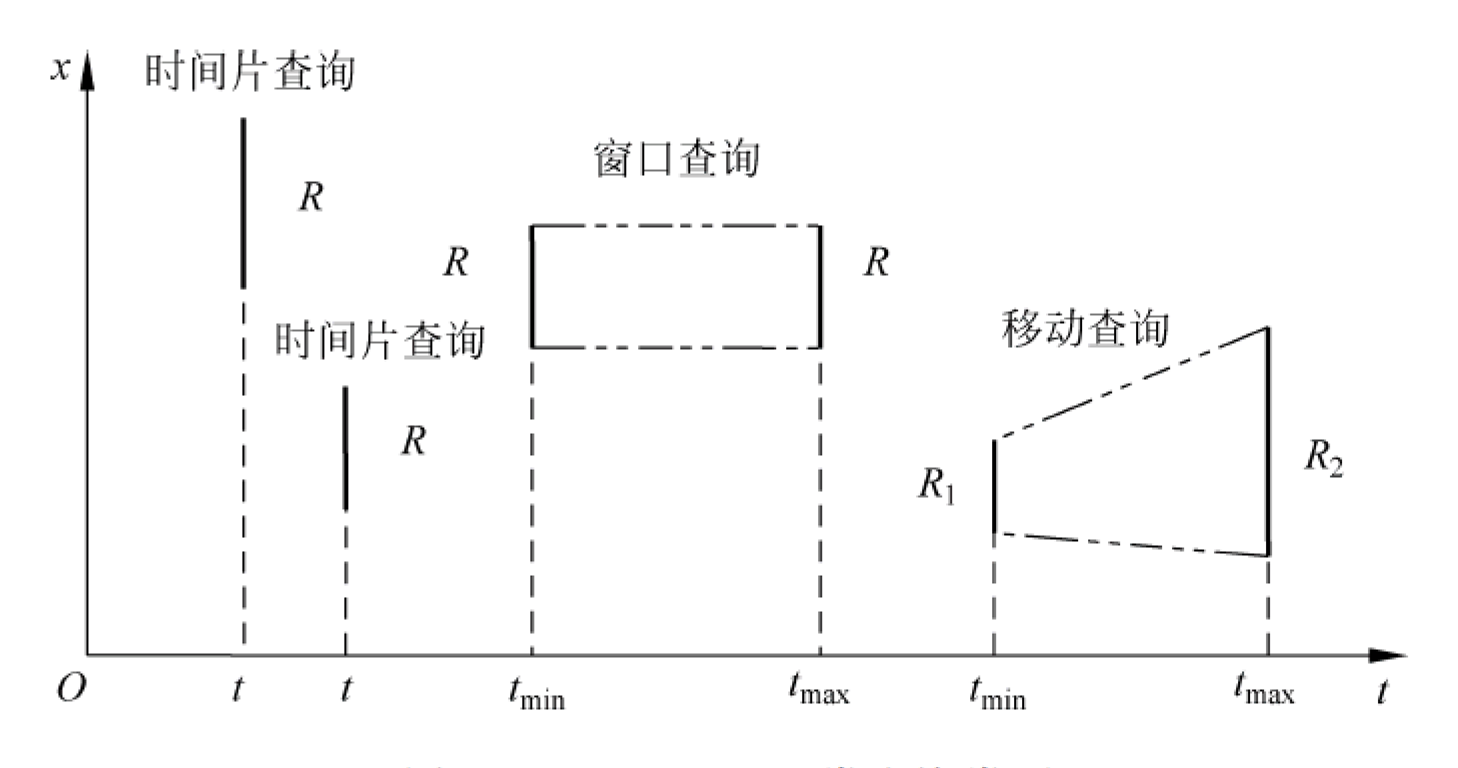


图 8-7　TPR-tree 三类查询类型

建立一个性能良好的 TPR-tree 需要考虑 3 个重要参数。

(1) 查询窗口 W：查询中设定的时间范围应该是能够较准确预见未来移动情况。若 iss(Q)表示查询 Q 的提出时刻，则对时间片查询，满足 $iss(Q)\leqslant t\leqslant iss(Q)+W$，而对窗口查询和移动查询，则有 $iss(Q)\leqslant t_{min}\leqslant t_{max}\leqslant iss(Q)+W$。

(2) 索引使用时间 U：索引建立之后能够有效使用的时间。

(3) 索引数据有效时限 H：索引使用时间与查询窗口之和。TPR-tree 须支持从 t_1 开始在时间范围 H 内的各类查询。

上述 3 个基本参数如图 8-8 所示。

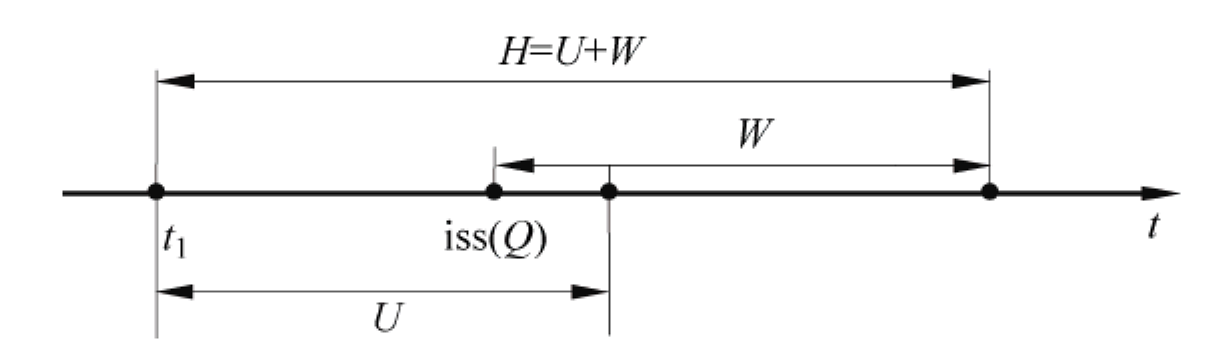


图 8-8　TPR-tree 3 个基本参数

3. 数据查询

TPR-tree 中 3 种查询类型实现算法可以分别讨论如下。

1) 时间片查询

首先计算在查询时刻各结点的 MBR，其次按照 R*-tree 查询算法进行查询。即在二维空间中，MBR 为(X_{min}，X_{max}))，VBR 为(V_{min}，V_{max})数据结点满足时间片查询 $Q=((a_{min},a_{max}),t)$ 的充要条件为$(a_{min}\leqslant X_{max}+V_{max}(t-t_1))\wedge(a_{max}\geqslant X_{min}+V_{min}(t-t_1))$。

2) 窗口查询

TPR-tree 中预测窗口查询和 R*-tree 类似，不同之处在于各结点 MBR 需在查询时刻动态计算。更新算法也与 R*-tree 相似，差异在于各度量值不同的计算方式。

(1) 设索引建立时间为 T_C，索引数据的有效时间为 H，$A(N,t)$和 $P(N,t)$分别为实体 N 在时刻 t 时的面积和周长，则实体 N 的面积和周长计算公式如下。

$$\text{实体 } N \text{ 面积} = \int_{T_C}^{T_C+H} A(N,t)\,\mathrm{d}t$$

$$实体\ N\ 周长 = \int_{T_C}^{T_C+H} P(N,t)\mathrm{d}t$$

(2) 设 $\mathrm{OVR}(N_1,N_2,t)$和 $\mathrm{Cdist}(N_1,N_2,t)$分别为实体 N_1 和 N_2 在时刻 t 的重叠区域和质心的距离，则实体 N_1 和 N_2 在重叠区域和质心之间距离计算公式如下。

$$N_1\ 和\ N_2\ 重叠区域 = \int_{T_C}^{T_C+H} \mathrm{OVR}(N_1,N_2,t)\mathrm{d}t$$

$$N_1\ 和\ N_2\ 质心间距离 = \int_{T_C}^{T_C+H} \mathrm{Cdist}(N_1,N_2,t)\mathrm{d}t$$

3) 移动查询

首先检测查询范围是否与该范围内 MBR 的移动范围相交。

设二维空间实体 $N=(\mathrm{MBR},\mathrm{VBR})=((X_{1\min},X_{1\max};\ X_{2\min},X_{2\max}),(V_{1\min},V_{1\max};\ V_{m2in},V_{2\max}))$，移动查询 $Q=(([a_{1\min},a_{1\max}],[a_{2\min},a_{2\max}],[w_{1\min},w_{1\max}],[w_{2\min},w_{2\max}]),t_{\min},t_{\max})$。

判断 N 和 Q 相交与否基本思想是，若两个多维移动矩形相交，则一定存在一个时间段，在此期间它们的范围在每一维空间上的投影都相交。两个移动矩形在第 $i(i=1,2)$维相交的时间段设为$[t_{j\min},t_{j\max}]\cap[t_{\min},t_{\max}]$。如果在两个维空间上相交的时间段没有同时交集，即 $t_1\cap t_2=\varnothing$，则说明这两个矩形不会相交，查询结果为空；否则返回相交的时间段。

4. 其他相关索引

TPR-tree 是一种具有代表性的移动对象数据索引方法，随后的研究者们也针对其中存在的不足提出了各种改进方案和新的索引方法。

(1) STAR-tree：通过引入自调整概念来提高 TPR-tree 的索引性能，它可有效处理基于时空对象现在或未来位置的各种不同查询，如区域查询、时间片查询和最近邻居查询等，并且还提供了在存储与查询性能之间和更新索引时间与响应查询时间之间较好的折中平衡。

(2) REXP-tree：通过假设移动点对象的位置在一个固定时间周期后将失效来索引移动点对象的现在与预测的未来轨迹。它修改了 TPR-tree 的插入与删除算法，并且使用"懒惰"更新技术来避免时空对象的运动信息长时间不能被更新而可能带来的问题。

(3) PR-tree：基本与 TPR-tree 类似，但它考虑用参数化矩形来表示时空对象的空间区域。每个参数化矩形都有一个时间间隔来表示时空对象运动的开始时间和结束时间，因此，一个时空对象在空间上用一个多边形来表示而不是连续的运动轨迹。

(4) NSI：使用一个沿着空间维的界限盒来表示本地空间的时空对象运动，并用一个多维的数据结构(如 R-tree)来索引该界限盒。但是界限盒的表示也会带来一定的误差。NSI 索引发生在初始空间中的运动，并且保护对象的局部性，但它对该索引树中的非叶子结点层的无效空间(Dead Space)进行了索引，从而减低了索引效率。

(5) VCI：引入 $V_{\max}$概念用来存储在该索引结构中每个结点所包含的所有对象中的最大允许速度。VCI 利用 $V_{\max}$延缓反映对象运动的更新索引代价和当对象移动时必须重新评估所有查询要求。VCI 优点是对于少量数目的查询具有良好的性能，而对于大量数目的并发查询性能却比较差。

(6) MP-tree：使用投影操作支持诸如时间片查询与区域查询等的查询操作。优点是，能有效处理分裂与查询操作，并具有高效的空间利用率。另外通过链表的使用，MP-tree 还可有效处理基于轨迹的查询。缺点是，当结点被输入时需花费更多的时间来进行投影操作的存储。

8.4.2 过去时间索引

按照前述基于过去时间的轨迹建模,移动对象的轨迹可以表示为一条折线。在建立基于 R-tree 索引时,叶结点包含轨迹线上各个线段的 MBR。而非叶结点即目录 MBR 的构造需要遵循"空间邻近性"原则,由此直接构造的 R 树会出现属于同一轨迹线的线段 MBR 可能位于相距较远的不同叶结点中,这会对"轨迹线查询"的效率产生不利影响。为此,需要在 R-tree 框架内,尽量做到同一轨迹线上的线段 MBR 具有存储上的"相近性"。索引的插入算法实际上也是构成索引的构造算法,而插入新的轨迹线最终归结为插入"线段 MBR",由此需要通过改进 R-tree 的插入算法及由此带来的分裂策略来构建适合于移动对象查询特性的新的索引结构。下面分别讨论两种比较典型的基于过去时间数据索引 STR-tree 和 TB-tree,它们的初始建构都与 R-tree 类似,而关键之处在于两者的插入算法都体现了叶结点中数据 MBR 的"同一轨迹线的相关性"。

1. STR-tree

STR-tree(Spatio-Temporal R-tree)是基于移动对象特性的时空 R-tree。相对于 R-tree 建立在最小扩展策略基础之上的插入算法,STR-tree 构建索引时除考虑常规的空间邻近性之外,还同时考虑保持轨迹线的部分完整性,尽量将属于同一个轨迹的线段保存在一起。

STR-tree 基本思想是,在插入新的线段 MBR 时,需要设计一个名为 FindNode 的查找算法,选择需插入线段所属轨迹线的上一个(前驱)线段所在的叶结点。当其前驱线段所在叶结点存在插入空间时,新线段就被插入,否则进行结点分裂操作。

1) 新线段插入

STR-tree 插入算法引入一个保存参数 p(Preservation Parameter)来表示保持轨迹完整性的层数。当返回的叶子结点为满时,检查 $p-1$ 层父结点是否为满。在图 8-9 中,$p=2$,所以只需要检查非叶子结点 1 层中的相应结点。

(1) 如果存在一个祖先结点具有插入空间,则对应叶结点就被分裂。

(2) 如果所有的 $p-1$ 中的祖先结点都无插入空间,算法使用 R-tree 中的选择子树算法在当前路径的右分支子树中查找满足条件的能够包含待插入线段的叶结点。

图 8-9 中阴影框表示包含所插入线段前驱线段的叶结点,而灰色框表示当前插入路径右子树上所有的结点。

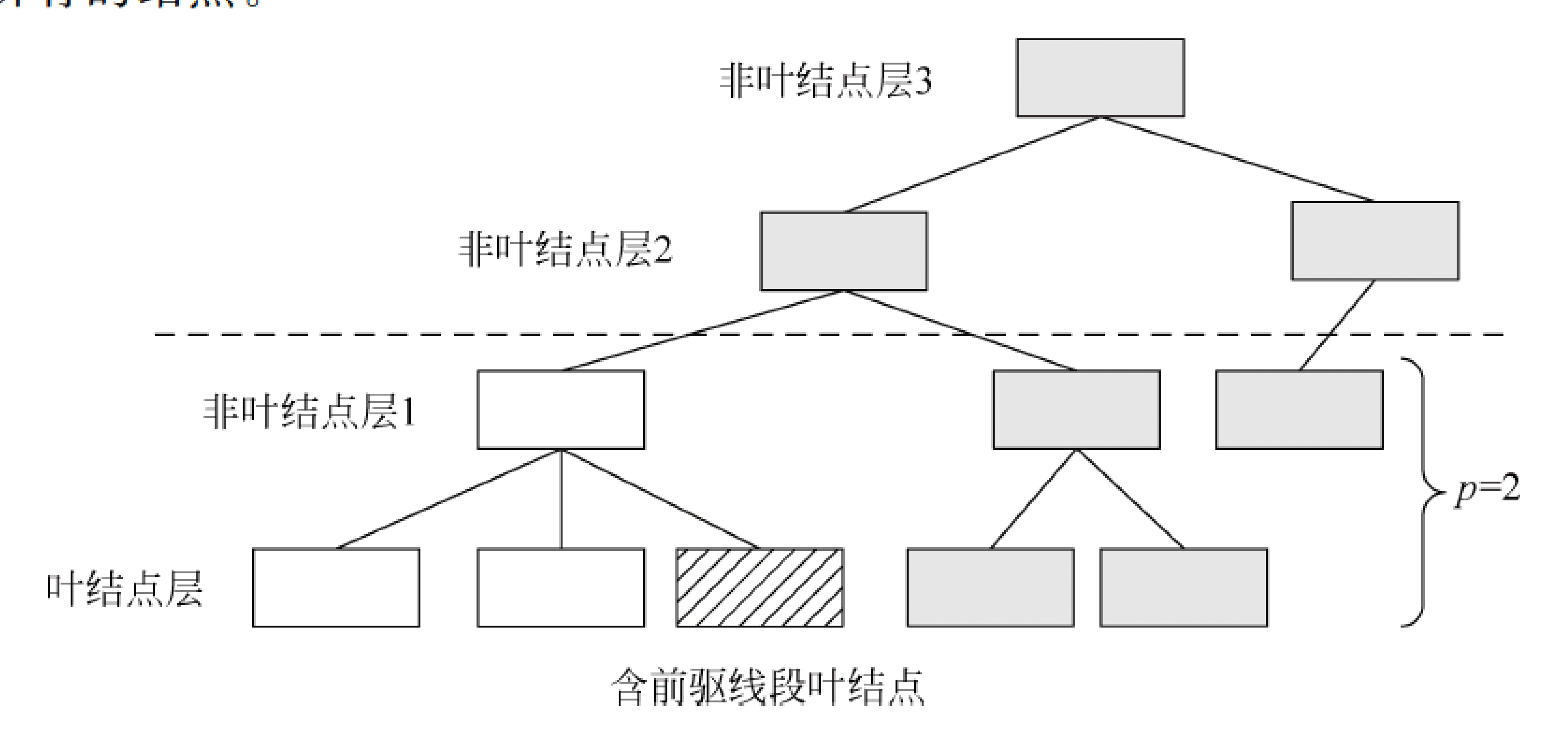


图 8-9 STR-tree 插入新线段

2）原有结点分裂

发生结点分裂时需要分别考虑叶结点和非叶结点两种情形。

（1）叶结点分裂。STR-tree 需要保持轨迹线的完整性，因此，分裂叶结点需要分析结点中包含哪种类型的线段。叶结点中任意两个线段可能属于或不属于同一条轨迹。当它们属于同一条轨迹时，可以有相同的端点也可以没有，但如果属于不同的轨迹则一定没有相同端点。因此叶结点中可能包含下面集中不同类型的线段。

① 不连续线段（disconnected segments）：与叶结点中其他线段均不连接。

② 向前连接线段（forward connected segments）：轨迹线中第一个线段。

③ 向后连接线段（backward connected segments）：轨迹线中最后一个线段。

④ 双向连接线段（bi-connected segments）：两个端点分别与同一轨迹线中的另外两条线段端点相连的线段。

STR-tree 叶结点分裂的基本思想是将最新也是最近的线段放置到新结点中。这使得新的线段比包含在原有结点的线段更有可能插入到新结点中。这种可能性可放宽对最小结点容量 m 的限制。

上述几种线段类型分割策略如图 8-10 所示，其中实线段表示新插入线段，虚线段表示叶结点中原有线段。

① 若所有线段都不相连，由 QuadraticSplit 算法决定分裂，如图 8-10(a)所示。

② 若至少有一条线段为非连接，将非连接线段放置到新创建结点中，如图 8-10(b)所示。

③ 若没有不连接线段，最近向后连接的线段被放置到新创建的结点中，如图 8-10(c)所示。

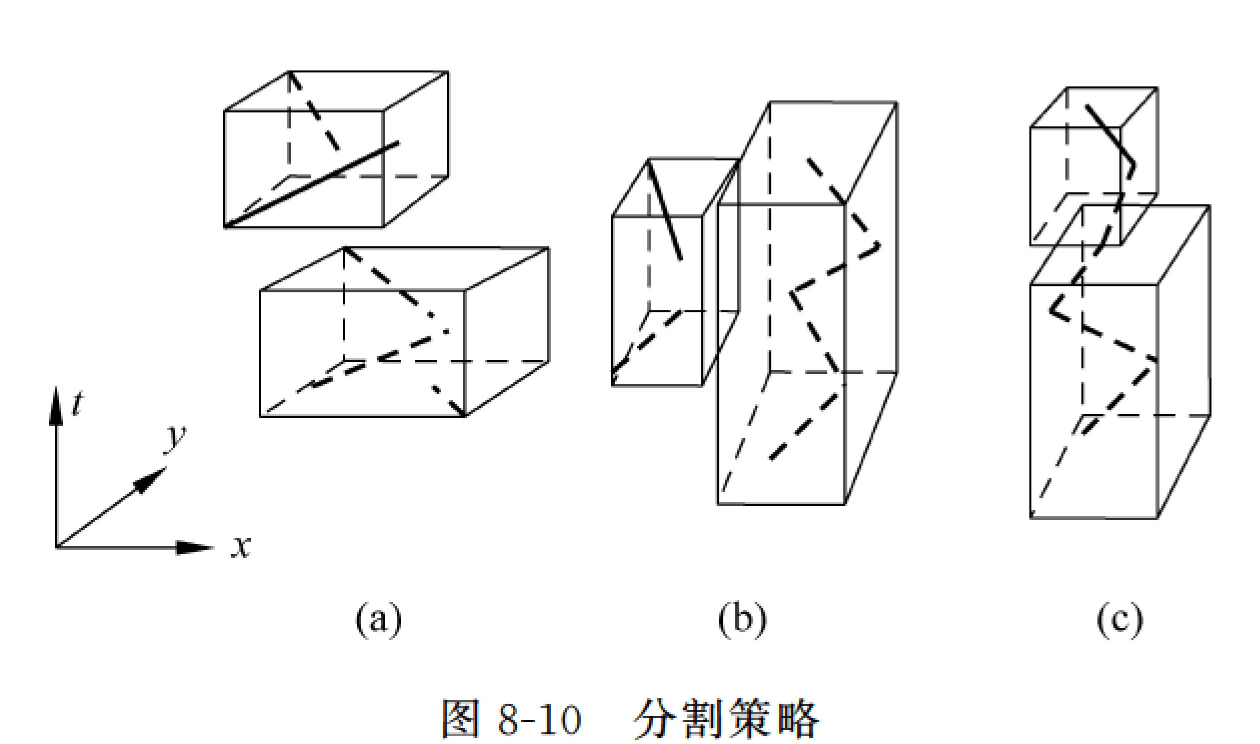


图 8-10　分割策略

（2）非叶结点分裂。STR-tree 的非叶结点较为简单，只需为新记录项创建一个新结点即可。使用这种插入和分割策略，使索引结构在保存轨迹的同时还考虑到了时间维情形。

2. TB-tree

STR-tree 索引对于窗口查询比较有效，但对于整体轨迹线查询却不够灵活。因为其线段 MBR 是否属于同一轨迹的信息在数据存储层面上是“隐含”的。为将这种信息“显式”化，需要引入 TB-tree（Time B-tree）索引结构。

1）TB-tree 基本思想

与 R-tree 构建相同，TB-tree 按照空间邻近性原则通过各个不同的目录 MBR 而将同一个轨迹分成了很多轨迹片段，而倒数第二层目录 MBR 包含的每一个 MBR 中的线段 MBR

就构成了一个叶结点。这样,一条轨迹的各个片段就可能会被存放在不同叶结点中。为了在数据存储层面显式表示同一条轨迹线上的线段 MBR,借鉴 B^{+}-tree 做法,TB-tree 就在叶结点层将同一条轨迹线的线段 MBR 通过双向链表链接起来。由此,TB-tree 保持了 R-tree 结构中线段对象的轨迹线特性,更适合于处理基于轨迹线的复杂查询类型。但其不足是空间中临近的不同轨迹线片段被存储在不同的结点中。

在插入更新过程中,TB-tree 自左向右逐步增长。最左边的叶结点是最早插入的结点,最右的叶结点最后插入。TB-tree 的每一叶结点包括部分轨迹,即轨迹分布在不相连接的叶结点中,因此在查询处理时需要设立基于轨迹标识符检索段。具体做法是,通过有层次的数据结构连接叶子结点。使用双链表按照时间顺序把包含相同轨迹部分的叶结点连接起来,这样达到保留轨迹的目的。

TB-tree 结构基本思想如图 8-11 所示,其中图 8-11(a)表示一条轨迹,图 8-11(b)表示在 TB-tree 索引结构中的对应部分,属于同一条轨迹的首尾端点相连的线段 $C1$、$C3$、$C5$、$C7$ 和 $C8$ 等在叶结点中通过适当指针方式(如双向链表)相互链接。

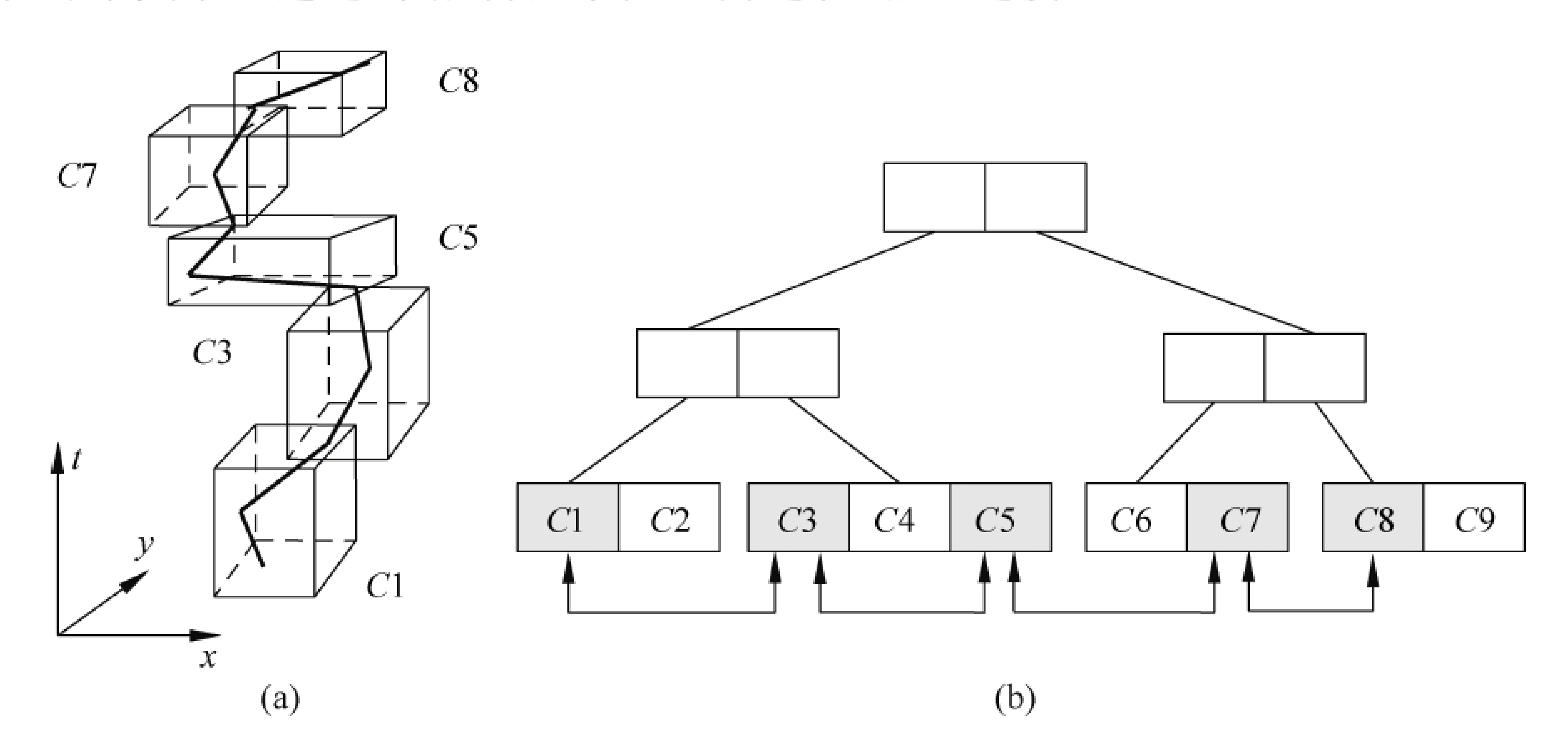


图 8-11 TB-tree 结构基本思想

2) TB-tree 插入更新

TB-tree 将各个移动对象的轨迹线切成许多片段,每一段包括 M 条线段,M 也称为扇出,表示每一叶结点包括 M 条轨迹片段。插入过程如图 8-12 所示,整个过程的重要阶段是图中的标号①～⑥。在插入算法中,当插入新数据时,须查找新数据所属轨迹所在的叶结点。

遍历 TB-tree 时,从根结点开始,依次进入与新轨迹片段 MBB 相交的每一叶结点,并选择包括与新片段相连接线段的叶结点(图 8-12 阶段①)。其中的 FideNode 算法具有与 STR-tree 相同的搜索过程。如果叶结点已满,需执行分裂算法。叶结点的分裂将影响完全保留轨迹的原则。因此,可以创建一个新的叶结点。在图 8-12 中,向上逐渐寻找未满的非叶结点(从阶段②～④)。选择最右边路径(阶段⑤)来插入新的结点。父结点中(阶段⑤)如果有空间,则插入新的叶结点。如果父结点已满,通过在非叶结点层 1 上创建一个新结点来分裂父结点,新创建结点有一个新的叶结点作为唯一子孙,必要时向上传播分裂。TB-tree 的增长从左到右,即插入时首先是最左边的,最后是最右边的。

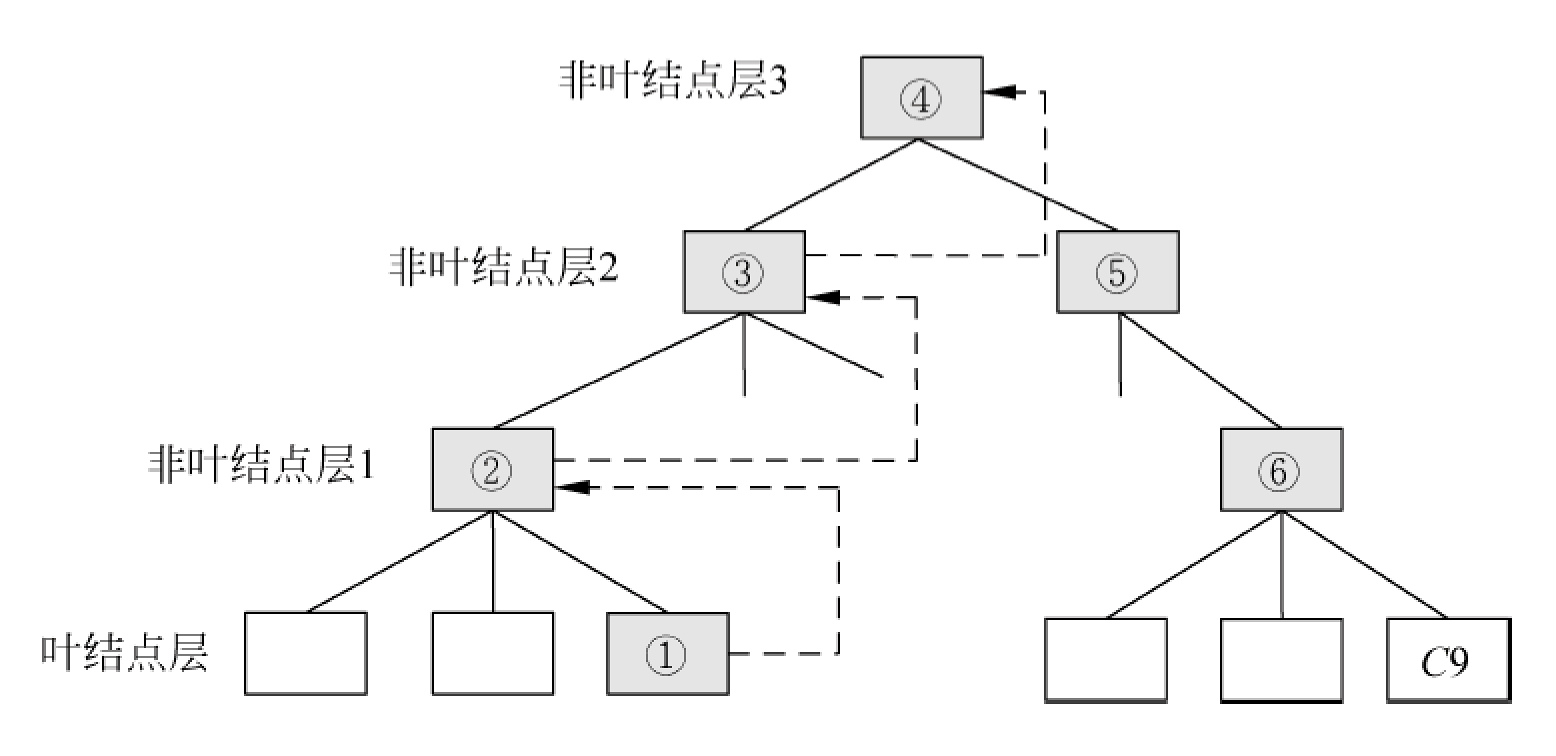


图 8-12　TB-tree 的插入过程

8.5　路网移动对象数据索引

MOD 中移动对象按照所在运动空间状态可分为下述 3 种情形。

(1) 非受限运动。例如,战争状态下飞机在天空中的飞行、一定范围内军舰船舶在海洋上的航行及巡航导弹的飞行等,其特征是移动对象的运动路线没有限制,可在各个方向上自由运动。

(2) 受限运动。例如,船舶在海洋上的航行会受到礁石和冰山等的限制,人在荒野中行走受到悬崖峭壁及河流沟壑的限制等,其特征是移动对象的运动受到固定物体的阻碍,但在这些阻碍物之外运动不受限制。如图 8-13(a)所示的在城市街区情形。

(3) 路网运动。例如,汽车在高速公路网络中运动、火车在轨道网络中运行、飞机和船舶在确定航线中运动和弹道导弹的飞行等,其特征是移动对象的运动轨迹需要预先设定,一旦设定后,对象的运动过程就不能再在给定道路或航线网络之外进行,如图 8-13(b)所示。

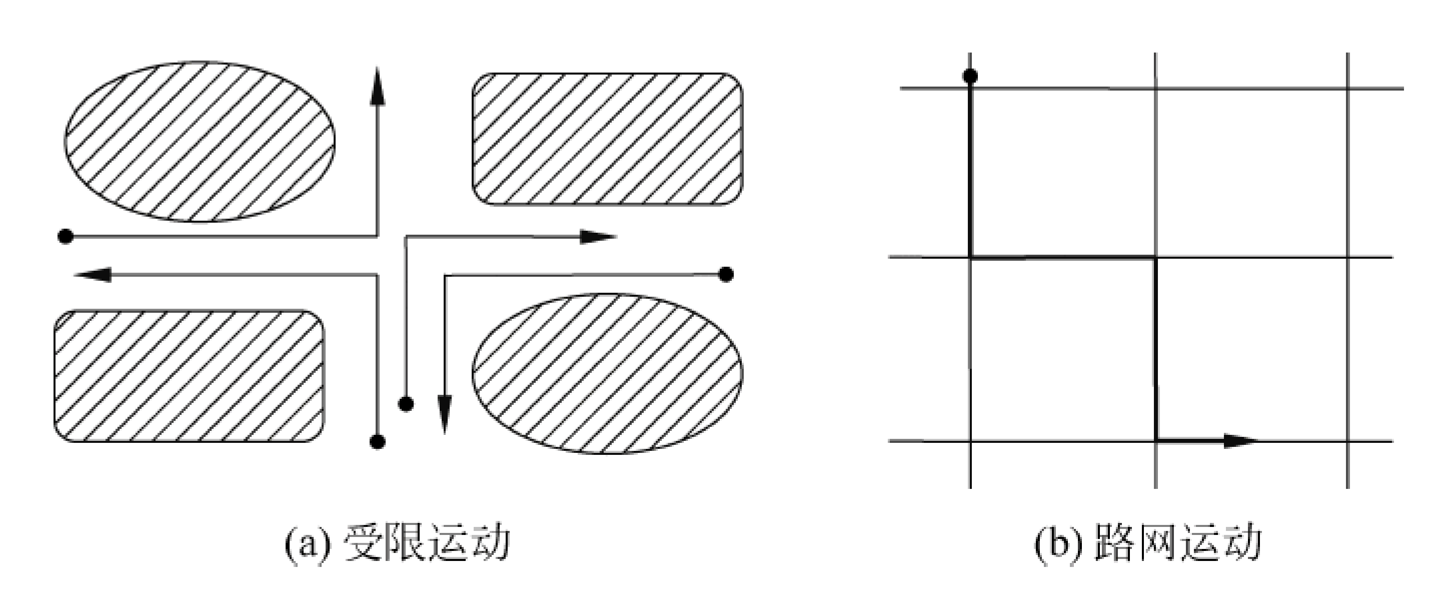


图 8-13　受限运动和路网运动

第(1)种是 MOD 研究的一般情形,MOD 领域中前期的大部分工作都可以用来处理此类问题。

第(2)种可以通过适当步骤转换为无限制的情形,如通过某种划分将所给出的查询窗口分解为多个更小部分,而每个小查询窗口成为无限制情形,从而使用各类已有的技术与方法。

第(3)种中的路网(Road Network)在研究中实际上是一种较为新颖的情形,它可以通过适当方式将一般 MOD 中的"两个"处理维度上的移动转换为"较低"处理维度上的移动,

即将需要同时考察"轨迹线"变化和"移动对象"变化的情形转换为只需考虑"移动对象"变化的情形。现实中大多数移动对象的运动网络空间都可近似看作成路网模型。此时,确定移动对象位置无须使用精确二维空间坐标,只需要记录路网中线性参考坐标即可,正如相关文献中所讲,路网模型中移动对象位置信息实现了从二维空间数据降到"1.5 维空间"数据。实际上,在交通运行工具与技术不断发展进步的当今世界,移动对象的路网运动可以看作是一种更为普遍的形式,应用中的大部分移动对象课题都可基于给定路网展开。近年来,基于路网的移动对象数据管理已经成为 MOD 技术研发的具有价值和应用前景的热点课题。

8.5.1 路网模型

现实生活中路网结构可能相当复杂,计算机难以直接描述和处理其相应状态,一般都是借助数学模型来近似表示实际路网。一个地区的国道与省道路网及其相应数学模型如图 8-14 所示。

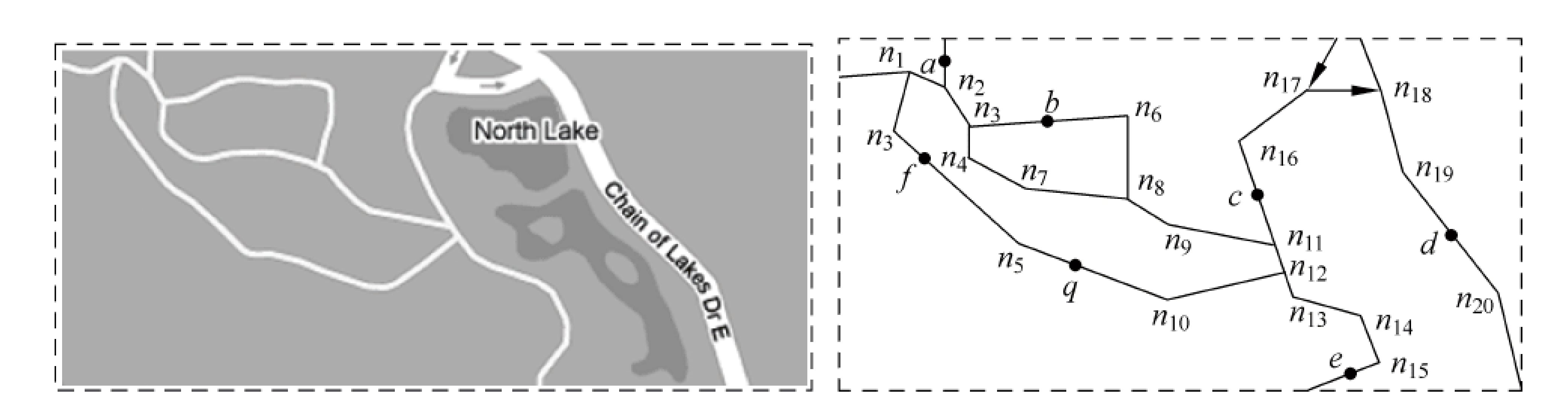


图 8-14 城市区域道路的路网模型

从数据管理角度考虑,现有路网模型主要分为静态路网模型和动态路网模型两大类。由于动态路网模型涉及当前道路的众多不定因素给研究带来了一定难度,目前关于此类模型仍处于建模研究阶段,而静态路网模型相对简单且容易实现。

现实生活中,路网中各种道路与路段长短不一、形状各异,道路在城乡之间的分布也不均衡。通常城市交通流量大、道路网络密集,而乡村移动对象相对较少、道路网络稀疏。此外,道路之间还存在相互穿插等复杂情形。但路网在结构上相对稳定,改动概率较低,更新周期较长,可以认为其在整体上会长时间保持静态。因此,目前路网移动对象数据管理技术大多都是基于静态路网模型提出的。根据道路抽象方法不同,静态路网模型又可以分为面向路段(Segment)路网模型和面向路径(Road)路网模型。

1. 面向路段路网模型

路网从图论观点可以看作是一个无向连通图的图形结构。面向路段的路网模型是从图论观点出发,将路网建模成为"交点(结点)"集合与"路段"集合组成的二元组。

定义面向路段的路网建模相应的路网模型 $G=(N,E)$。其中,N 表示这些"交点"(Node)全体的集合,这里的"交点"包括路网中各个线路的起始点、终点和交叉点;E 表示路段全体的集合,而路段(Edge)是线路中两相邻交点之间的部分。

面向路段的路网模型示例如图 8-15 所示,其中 n 表示交点,e 表示路段。

关于基于路段的路网模型需要注意下述问题。

(1) 模型中交点 $n\in N$ 的位置表示为二维点坐标 $P_n=(x,y)$,路段 e 连接给定线路上两

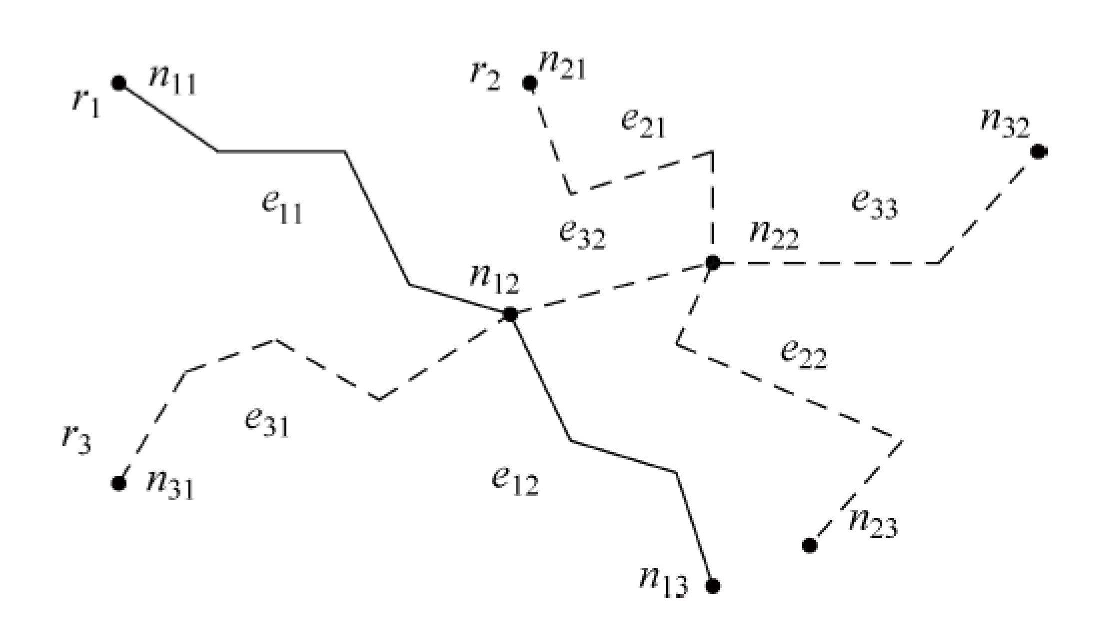


图 8-15　面向路段的路网模型示例

个相邻交点。

(2) 移动对象在路段 e 上轨迹表示为折线 $T_e=(P_1,\cdots,P_k)$，k 表示移动对象更新的次数。

(3) 参数 pos∈[0,1]表示移动对象在此路段中的位置，当 pos=0 表示移动对象位于路段起始点，pos=1 表示位于路段终点；而 pos∈(0,1)表示正在该路段运行。

(4) 在上述约定之下，移动对象在路段 e 中的位置可表示为 $D(G)=E\times \text{pos}$，而移动对象的轨迹函数表示为 $f: T\rightarrow D(G)$，其中，T 是相关时间域。

(5) 面向路段的索引模型简单明确，但它分割了道路的自然特征，对移动对象的全路段索引性能不高，并且按照一般直觉习惯，并不认为两个路段有交点就是一条路，而更为认可路口是两条或多条道路的交叉点。

2. 面向路径路网模型

面向路径的路网模型为二元组：

$$G=(R,J)$$

其中，R 表示路网中所有路径(Road)的集合；J 表示 R 中各个路径的交点(Junctions)集合。

面向路径的路网模型实际上是将路网看作是路径及它们之间交点的集合。

面向路径的路网模型示例如图 8-16 所示，其中，方结点表示路径端点，圆结点表示路径相交点。

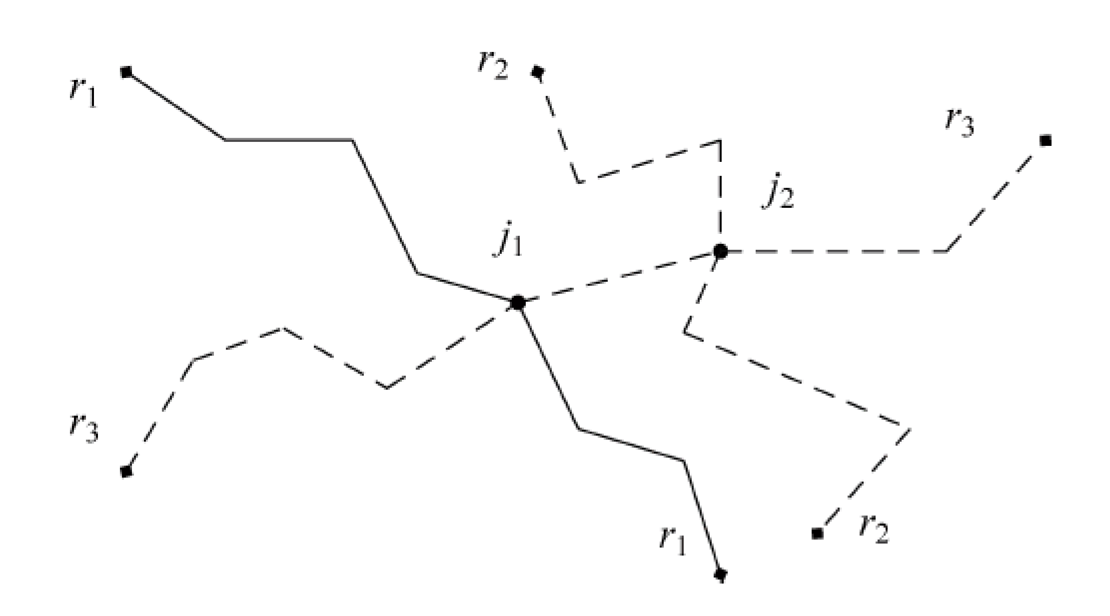


图 8-16　面向路径的路网模型示例

关于面向路径的路网模型需要说明下述各点。

(1) 在模型中，$r\in R$ 表示为由多个直线段相互连接的折线并记为 $l_r=(p_1,\cdots,p_k)$，其中 $P_i=(x_i,y_i)(1\leqslant i\leqslant k)$，$k$ 为路径的度量。

(2) 引入参数 pos：当 pos=0 表示移动对象位于路径起始点，pos=1 表示位于路径终



点，而 pos∈(0,1)表示正在该路径运行。

(3) 移动对象在路径 e 中的位置可表示为 $D(G)=E\times \text{pos}$，而移动对象的轨迹线函数可以表示为 $f: T \to D(G)$，其中，T 是相关时间域。

(4) 面向路径模型比面向路段模型减少数据的存储量，但是不利于管理运动在道路上的移动对象，影响移动对象数据的更新效率。

索引是时空数据库的基本查询技术，建立在面向路段和面向路径路网模型上的路网移动对象数据管理也不例外。大多数静态路网模型的索引结构都是借鉴 R-tree 或 R^*-tree，下面简要介绍两种具有代表性的路网索引技术，即 FNR-tree 和 MON-tree。

8.5.2 面向路段移动对象索引 FNR-tree

FNR-tree(Fixed Network R-tree)是 Frentzo S 等人于 2003 年提出的一种管理路网移动对象的经典索引技术，也是目前移动对象研究和应用领域中使用和借鉴较多的索引技术，不少移动对象索引都是通过对 FNR-tree 结构和算法进行改进或增添辅助结构以提升某方面索引性能的，FNR-Tree 索引结构在处理路网移动对象索引方面具有代表意义。

1. FNR-tree 架构

FNR-tree 采用基于路段路网模型，具有索引路段的二维 R-tree(2DR-tree)和索引路段移动对象一维 R-tree(1DR-tree)森林的上下两层索引架构。

1) 上层 2DR-tree

上层 2DR-tree 索引给定路网中的各个路段，并按照如下定义其结点。

(1) 非叶结点项。数据结构为二元组 Nocle=(ptr, MBR)。

① ptr：指向子结点的指针。

② MBR：包围其所有子结点路段的二维最小限定矩形。

(2) 叶结点项。数据结构为二元组 Leaf =(LinelD, MBR, Orientation)。

① LinelD：路网中的路段标识符。

② MBR：包含路段的最小限定矩形。

③ Orientation∈{0,1}：路段在 MBR 中的位置标志。

每个叶结点都包含路网中的一条路段数据。

2) 下层 1DR-tree 森林

对于上层 2DR-tree 中的每个叶结点都建立一棵 1DR-tree 用以运行索引在叶结点存储路段上的移动对象数据。每棵 1DR-tree 中结点定义如下。

(1) 非叶结点项数。据结构为三元组 Node(ptr, Tentrance, Texit)。

① ptr：指向其子结点的指针。

② Tentrance：其子结点中所有移动对象轨迹记录的最小值。

③ Texit：其子结点中所有移动对象轨迹记录的最大值。

(2) 叶结点项数。据结构为四元组 Leaf=(MovingObjectID, TentrancTe, TexitT, Direction)。

① MovingObjectID：移动对象的唯一标识符。

② TentranceTe：移动对象进入此路段的时间。

③ TexitT：移动对象离开此路段的时间。

④ Direction∈{0,1}：表示移动对象运动方向，值为 0 时表示对象从道路左边进入，否则为 1。

FNR-tree 的两层 R-tree 索引架构如图 8-17 所示。

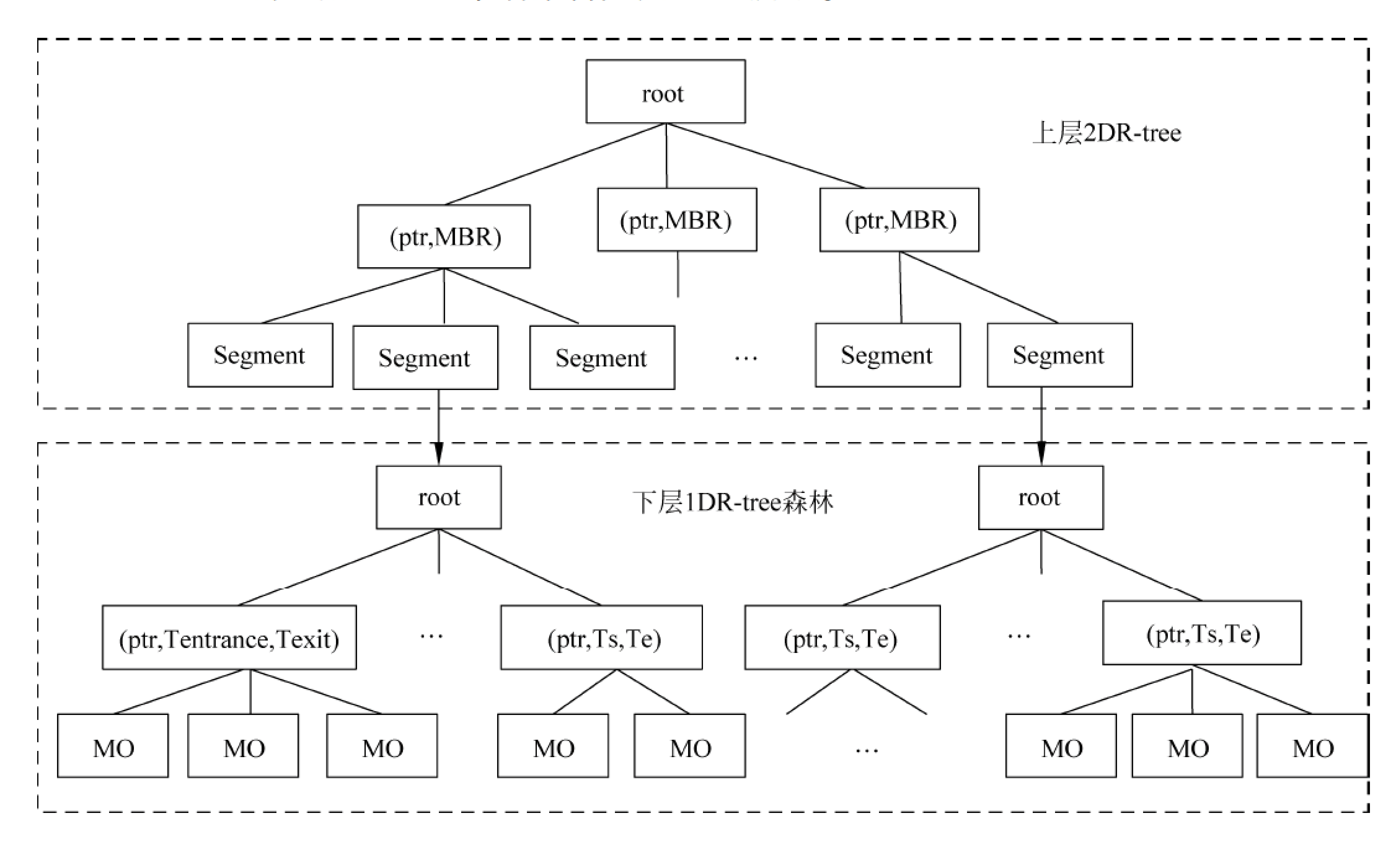


图 8-17　FNR-tree 的两层 R-tree 索引架构

2. FNR-tree 查询

R-tree 是一种高度平衡树且是一种支持完全动态的空间索引，其插入、删除和查询都可随时进行，不需要周期性地进行索引结构重组。FNR-tree 索引以 R-Tree 为基础，继承了 R-tree 的一些优良特性，利用两层 R-树结构将移动对象信息、空间信息和时间信息三者有机结合，具有较高空间利用率且在窗口查询方面有较好的性能。

1）窗口查询

基于 FNR-tree 的查询操作通过下述步骤实现。

(1) 在 2DR-tree 中搜索与空间查询窗口相交的线段，记录相应叶结点上路段 ID 和 R-tree 指针。

(2) 由指针进入相应 1DR-树执行 R-树搜索算法并记录其搜索结果所对应的 ID。

(3) 对(2)中 ID 与(1)中 ID 求交获得最终查询结果。

由于 R-tree 数据操作采用"自顶向下"搜索策略，因此 FNR-tree 每次查询或更新都将执行两次 R-Tree 的自顶向下搜索。例如，查询某给定矩形区域一段时间内移动对象时，首先要查询所有在该时间段内或与该区域相交的索引项，算法从 2DR-tree 根结点开始，自顶向下搜索所有 MBR 中与查询区域有关的数据项直到叶结点。其次根据 2DR-tree 叶子结点所对应的 1DR-tree，对 1DR-tree 再进行一次自顶向下过程以搜索所有与查询时间段相交的 1DR-tree 叶结点，最后返回所有相交叶结点移动对象信息，至此完成查询。查询算法可能还将搜索树中多个分支。

FNR-tree 在对移动对象精确匹配搜索过程中，可能需要从顶部同时向多个叶子方向遍

历，如果查询到 2DR-tree 中满足条件的多个叶结点，可能还要搜索所有与其对应的 1DR-tree，直到找到与其匹配的移动对象数据才能终止。因此，这种搜索方式会导致较大的结点 I/O 代价和 CPU 消耗，FNR-tree 查询性能就会受到一定影响。

2）路径查询

路径查询可表示为二元组 road_query=（road，t_1，t_2），其语义表示在[t_1，t_2]时间段内，行驶在路径 road 上的所有移动对象。

由于 FNR-tree 以路段为单位进行存储，并未记录路段与路段之间的关联信息。进行路径查询时，首先在上层 2DR-tree 中通过递归算法查找所有与查询路径有关的路段；找到路段后，再对满足条件路段所指向的下层 1DR-tree 进行搜索，通过时间窗口查找到满足查询条件的移动对象数据。此时，FNR-Tree 在路径查询中存在冗余重复搜索，索引结构查询效率有所降低。

在对过去轨迹查询过程中，需要遍历整棵上层 2DR-tree 和下层 1DR-tree 森林，导致查询性能随索引移动对象数量的增加，算法磁盘 I/O 次数成指数级增长。这说明，尽管在理论上 FNR-tree 可以支持移动对象历史轨迹查询，但实际应用中却不易实现这样的查询请求。

3. FNR-Tree 更新

FNR-tree 以路径路段为空间粒度构建索引，其中 2DR-tree 管理路网中路段信息，其中每个叶结点中包含一个指向其对应 1DR-tree 的指针。对应 1DR-tree 用于记录某个时间间隔内相应路段上的移动对象的运动信息。除非路网中线路发生变化，一般情况下对 2DR-tree 极少进行更新，而 1DR-tree 则会根据路网中移动对象的运动情况进行动态更新。FNR-tree 以路段为基本元素建立索引，这将导致当路段信息较为复杂时 2DR-tree 会产生大量叶结点，且当移动对象从一个路段进入到另一个路段时需要对底层 1DR-tree 进行大量的更新操作。

由上述讨论可知 FNR-tree 存在如下一些不足之处。

（1）不易查询历史轨迹：FNR-Tree 在进行历史轨迹和路径查询过程中需要搜索整个上层 2DR-tree 中的路段，可能出现大量无效路径查询，系统需付出较高查询代价。

（2）2DR-tree 过多叶结点：2DR-tree 每个叶结点仅包含一条路段，路网中大量的路段就会产生大量叶结点，这些叶结点中缺乏移动对象数目信息，使得移动对象流量查询实现困难。

（3）产生无效查询路径：FNR-tree 沿用 R-tree 自顶向下搜索策略，精确查询时可能需从顶部同时向多个叶子方向遍历，造成大量重复和无效的查找路径，消耗系统资源。

（4）路段间缺乏联系：FNR-tree 采用路段存储模式，缺少路段之间连接信息，在进行路径查询时需遍历上层 2DR-tree 以搜索查询路径有关的路段，对系统性能影响较大。

（5）实际处理不够细致：FNR-tree 将移动对象视为随机在路网中运动，为了简单将移动对象出现在路网每个位置的概率视为均等的。在现实路网中，不同路径上交通繁忙程度不同，有运动的和静止的、速度快的与速度慢的，相应移动对象的更新与查询请求的频率也不尽相同。由于 1DR-tree 只记录移动对象存在于路径的时间段，难以反映移动对象在路径中间停止运动或是改变方向的具体信息。

正是出于解决上述不足的需要，人们对 FNR-tree 提出了各类改进方案，MON-tree 就

是其中之一。

8.5.3 MON-tree

为克服 FNR-tree 的不足，Victor 等在 2004 年提出 MON-tree（Moving Objects in Networks Tree）。

1. MON-tree 架构

与 FNR-tree 相似，MON-tree 也是一个两层混合索引结构，由两层 2DR-tree 和沟通两层 2DR-tree 相互之间关联的 Hash 路径表组成。

1）上层 2DR-tree

MON-tree 上层由一棵 2DR-tree 和一个 Hash 表组成，其中，2DR-tree 针对路网拓扑结构建立索引，Hash 表用以建立上下层之间关联。

（1）2DR-Tree 非叶结点项。数据结构为二元组（MBR，childPtr）。

① MBR：包含所有子结点的最小限定矩形。

② childPtr：指向叶结点的指针。

（2）2DR-tree 叶结点项。数据结构为二元组（MBR，treePtr，polyPtr，ptr）。

① MBR：包含该条路径上所有移动对象的最小外接矩形，每个叶子结点只包含一条路径。

② treePtr：双向指针，指向路径哈希表。

③ polyPtr：指向路径实际存储的物理位置。

④ ptr：指向下层 R 树森林的指针，对应于该条路径上的移动对象。

（3）Hash 表。用于快速定位上层 R 树的叶子结点中的路径，Hash 表数据项结构为二元组（roadsid，treePtr）。

① roadsid：叶结点中路径的标识符。

② treePtr：指向上层 2DR-tree 叶子结点对应的路段。

2）下层 2DR-tree 森林

MON-tree 下层部分针对路网移动对象建立索引，形成一组索引时间信息的 2DR-tree 森林，用于构建移动对象位置信息。位置数据项的数据结构为二元组（(pos_1，pos_2)，(t_1，t_2)），实际上也可将该二元组看作是一个 MBR，pos_1、$pos_2 \in (0, 1)$，分别表示 t_1 和 t_2 时刻位置。

（1）2DR-tree 叶结点项。数据结构为一个二元组（MBR，roadsid，objid）。

① MBR：包围该对象的最小限定矩形。

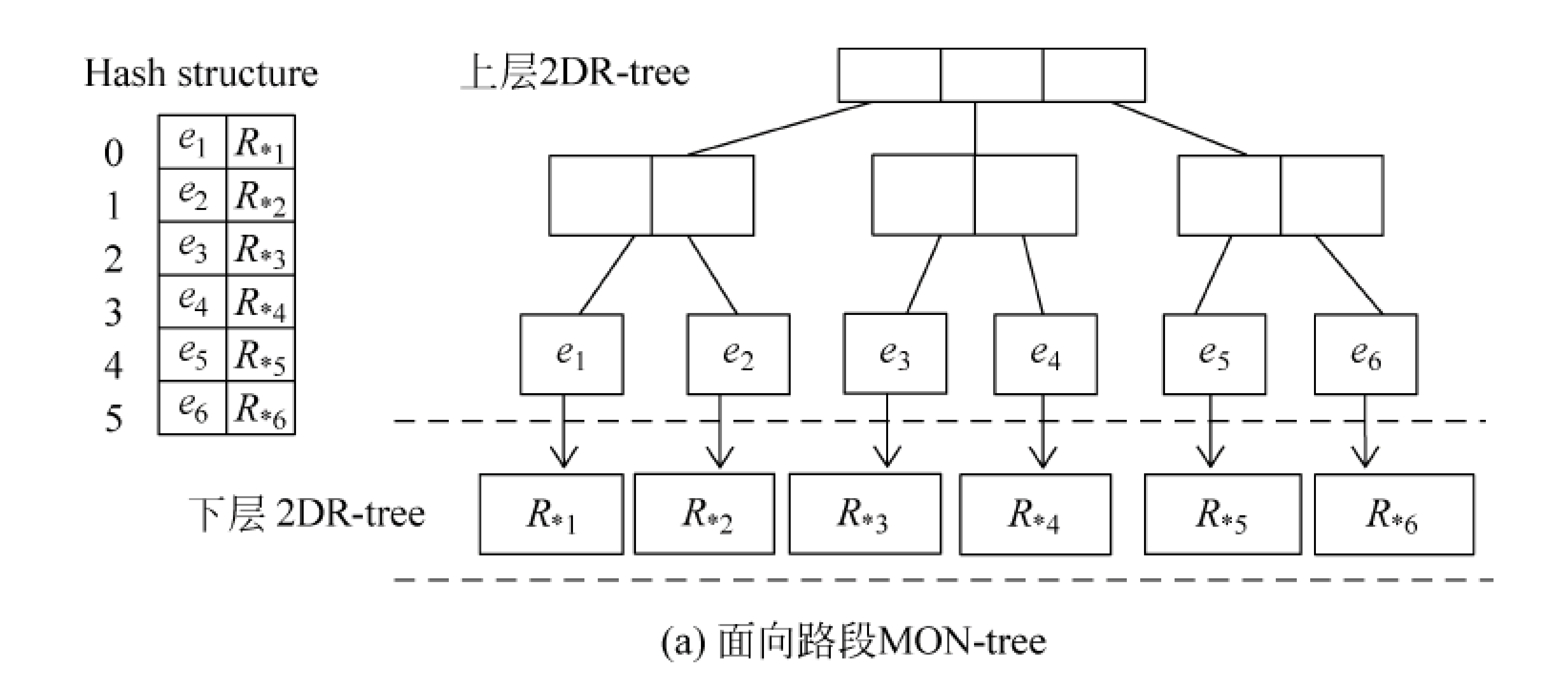
② roadsid：路径标识符。

③ objld：移动对象标识符。

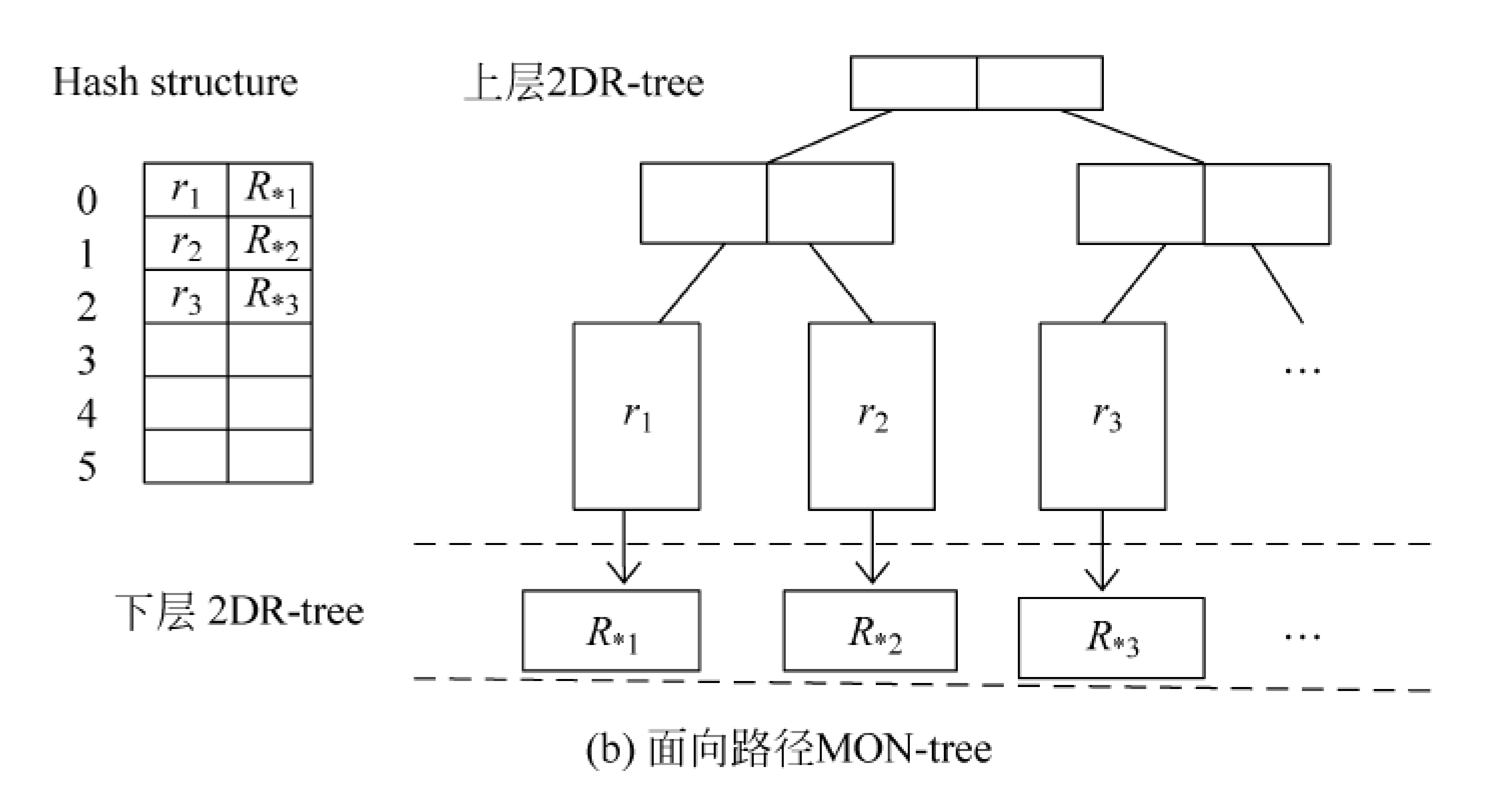
（2）非叶子结点数据项。与上层非叶子结点数据项类似且含义相同。

不同于 FNR-tree 只以路网中路段作为索引单位，MON-tree 可适用于静态路网模型中的两种模型，即面向路段和面向路径模型，对于不同应用而灵活选取所需要的数据模型。基于两种路网模型的 MON-tree 结构如图 8-18 所示。为了确定，以下讨论 MON-tree 时采用面向路径模型。

当 MON-tree 采用路径为索引单位时，能够减少叶子结点个数，储存数据比 FNR-tree 更为简洁，这样不仅减少了记录个数，而且降低了在表示移动对象跨越不同下层 2DR-tree



(a) 面向路段MON-tree



(b) 面向路径MON-tree

图 8-18　MON-tree 索引结构示例

时的工作量。实验表明,相对于 FNR-tree,MON-tree 具有更好的系统性能。MON-tree 采用面向路径模型时,可能会由于路径过长而产生死空间。

MON-tree 支持对移动对象的窗口查询和历史轨迹查询,但缺乏对时间信息管理和对网络拓扑空间的持续性优化,所以关于时间片的查询效率较低且不支持最近邻查询。

2. 窗口查询和插入更新

MON-tree 主要适合于窗口查询。由于需要记录历史数据,数据更新主要是插入操作。

1) 窗口查询

窗口查询(范围查询):输入查询窗口参数 $w=(p_1,p_2,t_1,t_2)=(x_1,x_2,y_1,y_2,t_1,t_2)$后相应查询步骤如下。

(1) 在上层 2DR-tree 中查询与 $r=(x_1,x_2,y_1,y_2)$相交的路径(折线)MBR,得到路径 ID。

(2) 根据得到的路径 ID,查询相应路径的存储数据(Polyline Representation),获取与 r 相交得到折线间隔(Polyline Intervals)的集合,并将这些间隔转换为查询窗口集合 $w'=\{(p_{11},p_{22},t_1,t_2),\cdots,(p_{n1},p_{n2},t_1,t_2)\}$,$n(\geqslant 1)$表示集合中窗口个数。

(3) 调用经典的 R-tree 查询算法,将得到的 w'中每个查询窗口对应的 MBR 在相应底层 2DR-tree 中进行查询匹配,以得到最终查询结果。

2) 插入更新

MON-tree 中数据插入存在如下两种情形。

(1) 路径插入。路径插入相对简单,只需向 Hash 表中直接加入线段 ID 即可,插入记录项为(polyId, null)。由于是新增路径,尚未存储移动对象在该路径上的运动信息,因此

该路径对应的下层 2DR-tree 指针为空。如果发生移动对象在该线段上运动,就为其构建下层 2DR-tree,修改相应空指针为指向该 2DR-tree 的实际指针,并插入该路径到顶层的 2DR-tree 中。如此可减小顶层 2DR-tree 的规模,避免对其查询时可能产生的不必要开销。

(2) 运动信息插入。插入记录项为(moId,polyId,p,t)。其中 moId 为移动对象 ID;polyId 为线段 ID;p 为移动对象在线段上的始点和终点且 $p=(p_1;\ p_2)$;t 为移动对象在线段上时间期间且 $t=(t_1,t_2)$。

首先,运动信息插入算法在 Hash 表中通过 polyId 搜索相应线段,如果线段对应的底层 2DR-tree 为 null,则新建一棵 2DR-tree,将 Hash 表中相应的 null 替换为指向该 R-tree 的指针,并将表示该线段的 MBR 插入顶层的 2DR-tree,其次通过经典 R-tree 插入算法将 STR(p_1, p_2, t_1,t_2)插入到新建的底层 2DR-tree。

本章小结

时空数据库在逻辑上可以看作是 SDB 和 TDB 的整合,主要研究形状和位置随着时间演进都发生改变的空间对象的存储与管理。从理论上考虑,MOD 属于时空数据库范畴,其基本特征是当时间变化时,重点考虑相应空间对象位置的改变。随着 GPS 技术的发展和移动通信终端设备的普及,产生了大量的只考虑位置信息而不考虑其自身形状的移动对象位置数据,由此就产生了有效管理移动对象数据的巨大应用需求,这就是移动对象数据库出现和发展的实际背景。

常规数据库存储和管理通常都是已经发生了进入“过去”范畴的数据,而移动对象管理的一个突出特征却是需要管理移动对象的当前位置数据和适当预测不久将来的位置数据,从而提供具有广泛实用需求的位置信息服务。这也成为现代数据管理技术的一项重要挑战。移动对象数据库中的核心技术大多都是围绕这一中心点展开的。

当然,管理移动对象“过去”位置数据信息对于移动对象数据库也必不可少,其基本思路就是对 R-tree 进行多种途径的技术拓展以适应新的情形。因此需要处理两个方面的问题:一是移动对象的移动轨迹,二是在给定路径上移动对象的位置。当前,对于给定路径上移动对象位置数据管理是领域研究的热点之一,也称为基于路网的移动对象数据管理。

路网移动对象数据管理的特点是可以通过引入相对距离函数而将涉及的空间维度由一般情况下的“二维”降低为“一维”,从而只需处理“二维时空”即可。从技术设计与实现角度,较低的维度相对于较高的维度具有更为明显和有效的优势。

与所有新型数据管理技术一样,由于还没有成熟的 DBMS,相应的数据查询主要都是依赖于相应的数据索引技术。例如,在时空数据库、XML 数据库和 MOD 中,数据索引技术始终是人们关注的重点之一。基于“当前”和“将来”的移动对象索引技术主要包括建立适当的运动轨迹(预测)函数和围绕函数处理的相应技术;基于“过去”的移动对象索引技术内核还是 R-tree,如路网移动对象数据索引的架构分为两层,上层使用经典 R-tree 索引路网数据,下层通常使用改进的 R-tree 索引移动对象数据信息。窗口查询和最近邻查询是移动对象数据最基本的数据查询形式。

主要参考文献

[1] 古廷，施耐德.移动对象数据库[M].金培权，岳丽华，译.北京：高等教育出版社，2009.
[2] 孟小峰.移动数据管理：概念与技术[M].北京：清华大学出版社，2009.
[3] 郝忠孝.移动对象数据库理论基础[M].北京：科学出版社，2012.
[4] 郝忠孝.时空数据库新理论[M].北京：科学出版社，2011.
[5] Dunham M H，Helal A. Mobile Computing and Databases：Anything New? [J]Acm Sigmod Record，1995，24(4)：5-9.
[6] Chen J，Meng X. Update-Efficient Indexing of Moving Objects in Road Networks[J]. GeoInformatica，2009，13(4)：397-424.
[7] Güting R H，de Almeida T，Ding Z. Modeling and Querying Moving Objects in Networks[J]. The VLDB Journal—The International Journal on Very Large Data Bases，2006，15(2)：165-190.
[8] De Almeida V T，Güting R H. Indexing the Trajectories of Moving Objects in Networks[J]. GeoInformatica，2005，9(1)：33-60.

第9章 大数据技术简述

随着计算机技术全面融入社会经济文化生活，信息增长已经达到了一个能够引发变革的崭新阶段。移动互联、社交网络、物联网和云计算等极大拓展了互联网的边界和应用范围，各种数据正在迅速增长的同时，其增长速度也在加快，这不仅使世界充斥着比以往更多的信息，而且数据内容越来越丰富，数据关系越来越复杂，以及数据更新频率也越来越高，由此创造出了"大数据"的新概念及其相关技术。如今，大数据几乎应用到了人们生产、工作和生活所涉及的各种行业、部门和领域中。

当前，大数据比较集中在基于互联网络的社会经济活动和人际交互活动，这就使得大数据与常规的数据概念有着基本的差异，也给数据管理新技术发展带来了挑战和机遇。实际上，大数据技术的出现使得已有的各类巨量数据成为有可能挖掘出更大潜在价值的软黄金资产，这种数据资产也许比其他固有资产更具有含金量。同时，大数据的影响并不局限于数据管理技术领域，因为一种新型技术可以对人们社会生活、经济活动及思维方式等方方面面产生广泛而深远的影响，从而有可能开辟一个崭新的时代。计算机技术开辟了信息时代，互联网技术开辟了网络时代，人们至少从计算机应用角度考虑而有理由相信，大数据技术或许有可能开辟出一个数据科学新时代。

本章简要介绍大数据的概念、大数据特征及应用，并从数据管理角度介绍基于大数据管理的数据库技术。

9.1 大数据基本概念

大数据是21世纪第一个十年过后发展起来的一种新型数据管理技术，其概念的内涵和外延仍在不断完善成熟的过程中。

大数据(big data)首先是一类数据，而数据本身就是一个难以精准定义的抽象的伞形概念。如果给数据挂上相对具体的实用场景，如从计算机存储管理数据视角审视数据，或许能够通过相关特征来对特定数据对象进行逻辑概括和技术界定。在实际应用过程中，基于计算机管理的"数据"可以具有3个不同的认知层面：数据集合的"数据自身"特征、数据管理的"技术处理"特征和数据使用的"领域应用"特征。其中，前两个特征属于计算机科学技术范畴，而后一个特征具有更多非计算机的领域刻画。

对于所熟知的RDB而言，其"数据自身"特征就是具有简单明了的平面表结构，基本数据对象(如元组)没有语义顺序强制要求，能够作为元素集合进行相对方便易行的存储管理等；其"技术处理"特征就是由于其建立在集合运算基础之上查询与更新机制，标准和完善的数据管理原理语言，能够实现非导航式查询和事务管理功能；其"领域应用"特征就是主

要用于商业、金融和服务业等行业内的事务管理业务。

大数据应当是比关系数据和对象数据等现有计算机实施有效管理的数据集合更为复杂的数据“类型”，为了比较清楚地厘清和把握其概念含义，通过“数据自身”“管理技术”和“领域应用”等计算机数据不同的认知层面或许是有所裨益的。

9.1.1 大数据自身组成特征

如前所述，关系数据主要来源于联机事务处理，对象数据主要着眼于多媒体数据、XML数据主要满足网络数据应用需求等。实际上，所涉及数据的来源就决定了其各类特征的基本因素。为了讨论大数据作为“数据本身”的特征，也需要始于大数据的实际来源。

1. 大数据基本来源

随着计算机网络技术的发展和移动通信设备的普及，社交网络、电子商务、物联网和云计算带给了人们在社会和经济生活形式方面的巨大变化，使得人们各项活动越来越依赖于计算机网络，也就是说，越来越网络化。这种由互联网连接起来的是一个由大量活动构件与多元参与者构成的生态系统，由终端设备、基础设施、网络服务提供者与接入服务提供商、数据服务提供零售商和数据服务使用者等一系列活动者共同构建。物联网、云计算和移动互联网框架平台，个人计算机、智能手机和平板电脑及各类传感器等承载方式等，共同构成了大数据各类来源的技术通道。上述数据的一个共同特点就是位于网络空间之中，人们通常将其称为网络空间中的数据资源，并将这种资源称为“数据界”。这也就是说明，大数据的主体主要来自“数据界”而不是借道于其他的途径获取。数据实际上来自于实体之间的交互，物和物的交互就会得到各类相互作用和关联的数据，人和物的交互就会得到各类测量和感知数据，人和人的交互就会得到各种社会交往数据等。网络时代使得物、人等实体之间的交互和彼此关联达到前所唯未有的广度和深度，由此产生的数据就构成大数据的基本源泉。

从应用技术角度考虑，主要有以下几种类型的大数据来源。

1）交易管理型数据

政务部门数据系统和企业内部数据系统等数据库数据，主要有政务管理信息、联机交易数据和联机分析数据等类型，其中包括人口普查数据、电子商务交易数据、线下实体店销售数据与分析数据等。这些数据记录过去发生的事项，需要完整精细地进行记录存储。作为结构化、静态和历史的数据，它们通常使用关系数据库进行管理和访问。

2）交互型数据

随着互联网和物联网等技术快速发展，交互型数据迅猛增长且具有多种类型。

（1）互联网上电子邮件、新闻、网络日志、微博微信、视频网站、通信及其他来源的社交媒体数据，包括视频、图片和文本等类型。

（2）物联网、移动设备和个人位置等各类传感器采集的数据，如设备和传感器信息、GPS和地理定位映射数据等。

（3）通过管理文件传输Manage File Transfer协议传送的海量图像文件数据，如天文望远镜拍摄的图像、视频数据和气象学卫星云图数据等。

交互数据主要由半结构化和非结构化数据组成，主要功能在于记录发生了哪种事项。

3）分析型数据

掌握和存储巨量数据的很多用户都会开始想方设法将“数据”转换为“资产”，从数据海

洋中分析和挖掘有价值的信息，为企业和单位行为提供科学合理的决策依据。因此，在数据处理过程中产生大量分析性数据。例如，淘宝会根据每个用户的个人购买与浏览记录，分析其购买习惯，与购买相同商品的用户进行相关性分析等，为每个用户定制个性化的推荐列表。这些数据正是大数据的价值所在，其特征是判断当前发生事项，预测将来又可能发生些什么。

2. 大数据自身特征

大数据的基本来源决定了大数据的各类基本特征。

1）数据类型多样性

由于网络空间已经涵盖了人和人的交互（互联网）、人和物及物和物的交互（物联网），因此大数据自身特性首先就是来源多样性以及带来的类型互异性。例如，基于环境生态的大数据包括地理地质、海洋气候、流行病与传染病、社会种族结构与社会经济生活等诸多来源的各类数据，因此这些数据的类型与描述格式也有很大差异，这些类型各异的情形对数据处理能力提出了更高的要求。

2）数据量级超大性

从理论上讲，网络空间能够触及地球村中所有的人和物，因此所产生的数据量必然巨大。2008 年世界科技界权威刊物 *Nature* 曾经在 2008 年出版了一期大数据专刊。在这期刊物的封面，除了如今广为人知的 big data 字样外，还特别在其后标上 science in the Petabyte era（科学处在 PB 时代）。正如其所预见的那样，现今大数据发展表明数据量已经进入到 PB 级别，通常人们也就认为这就是“大数据”的数量级标志。这种规模数据的存储与管理是常规数据库技术难承担的，由此也就构成了大数据有别于常规“海量”数据的量级特征。

3）数据价值低密度性

大数据通常需要长时间积累，因此组建大数据集合是一项费时、费力和花费大量资源的工程。虽然大数据是宝贵的财富资源，但在实际应用中，数据量的增加并不意味着数据价值显式的同步增长，因此超大量级的数据中相应的数据价值密度却可能很低，如在视频监控中，长时间连续不间断监控过程往往仅有一两秒的有用数据信息。如何通过强大的机器算法更迅速地完成全样本处理数据的价值“提纯”，已成为目前大数据背景下亟待解决的技术难题。这种数据高容量和价值低密度构成大数据集合又一有别于常规数据的基本特征。

4）价值实现的时效性

实时有效的数据查询是所有计算机数据管理技术的基本要求。大数据由于数据量巨大，相对于常规数据处理的实时性在大数据技术中往往需要放宽到处理结果的时效性，即在预期时间内获得大数据处理结果。由于社会生活和经济活动中人和人，人和物及物和物的交互都有时间期间的界定，超出了界定时间，交互就有可能毫无意义，因此如果不能在希望的时间之内完成大数据处理工作（如后所述，主要是决策支持等），即使大数据中存在很大价值，这种超预期时间的价值也就没有了任何意义。

9.1.2 大数据管理技术特征

大数据的来源确定了大数据不同于常规数据的自身特征，而在通过计算机管理大数据过程中，依据这些自身特征也就确定了大数据的各类技术特征。现今得到人们比较一致认

可的大数据概念实际上多是基于大数据自身特征而从技术处理特征层面所提出来的。例如，在 1997 年，当 Michael Cox 和 David Ellsworth 首次提出“大数据”概念时就指出，对于数据量大到内存、本地磁盘甚至远程磁盘都不能处理的一类数据可视化的问题称为大数据。这是首次将数据量级特性与数据技术结合而给出的相关定义。此后，经过人们的不断探讨和完善，现今通常接受和采用的是麦肯锡全球研究所(McKinsey Global Institute)在 2011 年给出的 4V 定义，即数据规模巨大(Volume)、数据类型众多(Variety)、数据处理时效性强(Velocity)和数据价值密度低(Value)。基于 4V 的大数据定义实际上就是大数据处理过程所应该具有的技术特征。

1. 大数据量技术处理特征(Volume)

伴随着社交网络、移动计算和传感器等新的渠道和技术的不断涌现和应用，世界上每时每刻都在产生过去难以想象的“很大很大”的数据量。计算机数据运算和储存以字节 B(byte)为单位，接下来依次增大使用的单位分别为 KB(Kilobyte)、MB(Megabyte 兆字节)、GB(Gigabyte，吉字节)、TB(Trillionbyte，太字节)、PB(Petbyte，拍字节)、EB(Exabyte，艾字节)、ZB(Zettabyte，泽字节)和 YB(Yottabyte，尧字节)，这些数据单位序列的换算关系是后者分别为前者的 $2^{10}=1024$ 倍。

根据统计，进入 21 世纪后的第二个十年，各个应用单位计算机系统实际数据量已从 TB 级分别跃升到 PB(1PB＝1024TB)、EB(1EB＝1024PB)乃至 ZB(1ZB＝1024EB)级。对于雇员人数超过 1000 人的几乎所有美国企业，自身的数据存储量大都超过了 200TB，而且不少企业平均还达到了 PB 级别。全球企业在硬盘上的数据存储量早已超过了 7EB。2010 年欧洲组织的存储总量约为 11EB，整个美国数据存储总量约为 16EB。PC 和笔记本电脑设备上个人存储量超过了 6EB。国际数据资讯公司(IDC)研究结果明，2009 年全球产生数据量为 0.8ZB，2010 年增长到 1.2ZB，2011 年已达到 1.82ZB，这相当于全球每人产生 200GB 以上的数据。与此相对比，美国国会图书馆 2011 年存储的数据约为 1/4000 EB。IBM 的研究以此据称，整个人类文明所获得的全部数据中有 90%是过去两年内产生的。根据 IDC 监测，全球数据量大约每两年翻一番，预计到 2020 年，全球将拥有 35ZB 的数据量，数据规模将达到当今的 44 倍。

在实际应用中，作为数据资源，人们当然不能凭空就可获得相应的大数据。问题在于，即使具有数据理论上的占有权或者得到了数据获取许可，面对如此巨大的数据，如何才能将其或者将其与己相关部分收入囊中呢？况且，由于相关成功的案例和数据的资源性质日益昭然，从人们的主观意愿来看，总是占有的资源越多越好，或者至少要比别人知道的数据更多，占有的数据更广。由此，如何获得以 PB 级别为目标的数据就成为大数据技术首先需要解决的问题。实际已经证明，使用常规数据获取方法难以得到 PB 级别数据，那就需要新的以搜索、爬取和高速下载为特征的大数据获取技术。

2. 多种类型技术处理特征(Variety)

如此数量庞大和快速增长的大数据，如前所述，不是只具有单一的来源渠道，而是呈现出多样化的显著特征。大数据并不简单地只是“量”的巨大和爆炸性的增长，而是由于数据来源渠道不同带来数据类型的多种形式，由此形成了大数据的第二个基本特征：数据类型多样性，实际上也就产生了计算机存储管理方面的巨大挑战。应用中选定的数据库大多都只能存储管理一种结构类型的数据，如 RDB 就只能存储管理表示为结构化的二维关系表数

据等。大数据来源于互联网和各类传感器等,这些大多都是半结构或非结构化数据。现有统计表明,全球结构化数据年增长率只有32%。而非结构化数据年增长率却为63%。在2012年,非结构化数据在整个互联网数据中的占比已经超过75%,而其中85%以上的数据都是音频、视频数据和网页数据等。巨量多来源数据的存储管理、跨界域数据的访问与计算等是此时有别于常规技术的大数据技术挑战,当前大数据存储相关技术主要有分布式文件系统(基于Hadoop框架的HDFS)、NoSQL数据库、虚拟存储和云存储技术等。

3. 时效性技术处理特征(Velocity)

大数据的"大",除了数据量在日益增大外,还表现为数据量增长的速度也在逐年增大,即大数据增长和收集的速度越来越快。互联网中连接的设备越来越多,随时都有越来越多的人在进行网上交易、发帖跟帖和上传音频视频等。同时由于通信技术的发展,数据收集速度也越来越快,即数据在网络中的流动加快。由此就有所谓的"一秒钟定律"之说,即在现实网络空间中,人们实际上是在完成着如下事项。

每一秒钟,发送290万封电子邮件。

每一分钟,向YouTube上传60小时的视频。

每一天,在Twitter上推送1.9亿条微博,并且发出3.44亿条消息。

每一天,在Facebook上发出40亿条信息。

注意到网络空间实际上使得人与人、人与物和物与物交互范围的倍增,而交互响应需要有时间范围的界定,也就是说数据的产生频率增大实际上也必然会有数据使用频率的增大,这样,网络空间中,从数据生成到数据使用或数据消耗,相应的时间窗口会变小或更为严格,即可以用于接收到数据信息的反应(如生成决策)时段将有严格的限定。获取数据到做出决策需要使用数据分析与数据挖掘相关技术,由于前述大数据的时效性特点,相应的技术与非大数据环境中的也会有明显区别和本质不同,由此需要新的更有效的与"时效性"相适应的大数据技术。

4. 低价值密度技术处理特征(Value)

大数据蕴含大价值,但却是犹如大海中拥有大宝藏。由于大海之大,从中获取宝藏通常被人们看作是"大海捞针",这也就是说,相对于大海的体量,大海中宝藏就具有"低密度性"。在实际问题中,如果数据没有价值或还有更大价值的其他数据,那么人们就不会关注其价值获取技术;如果数据量大,而大部分数据都具有明显的价值指向,价值提取没有新的技术难度,那么可以考虑使用或借鉴常规数据处理技术,这时数据尽管量大,但应该不属于人们所说的"大数据"范畴;如果大数据量大而其中价值密度很低且又相当"隐秘",难以通过常规方法进行提取,这就需要创建与之相应的大数据技术。此时主要是各类的数据分析与数据挖掘技术。但就当前状况而言,面对PB级别以上的低价值密度数据,还缺少有效的数据分析处理和数据挖掘算法,同时也没有理想的相关软件操作工具。如何通过强大的机器算法和高级分析技术更迅速地完成全样本处理的数据的价值"提纯",已成为大数据中亟待解决的技术难题。

大数据4V技术特征如图9-1所示。

通过上述分析,也可初步了解"大数据"与常用的"海量数据"联系与区别。实际上,大数据包含了海量数据,更超越了海量数据的原有内涵。体量巨大的结构化数据可以看作常规意义下的海量数据,而大数据包含更多的是半结构化和非结构化的数据,同时带来复杂类型

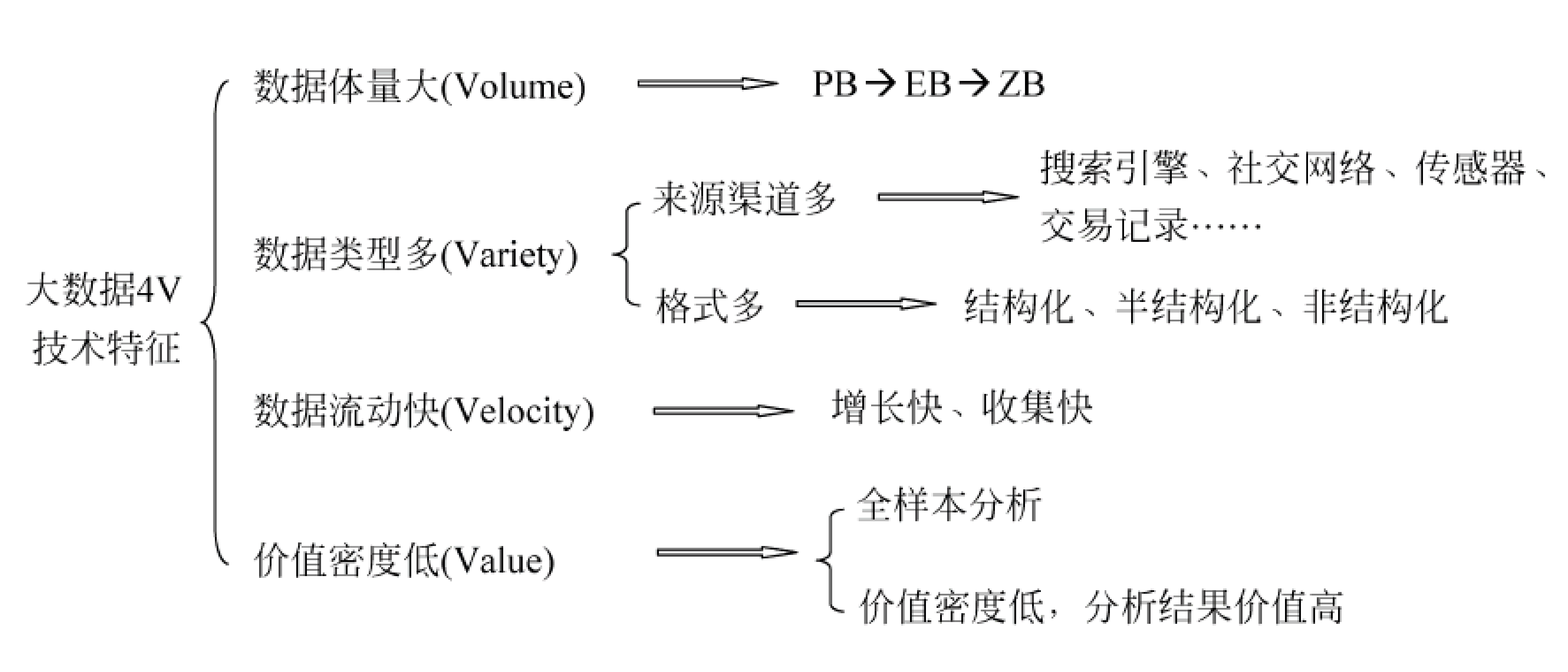


图 9-1　大数据 4V 技术特征

的数据处理方法。海量数据并不一定都有上述的 4V 特征。对于海量的结构化数据，管理和应用技术在逻辑上来说比较“单一”，用户通过购买容量更大的存储设备和处理速度更快的机器装置等就可提高相应的系统效率；但对大数据而言，其数据规模和类型复杂程度都超出常用设备技术按照合理的成本和时限捕捉、管理及处理的能力，因此形成了一个新型的更具挑战性的数据管理领域。

9.1.3　大数据领域应用特征

由大数据来源而产生的自身特征确定了大数据管理的技术特征，而无论人们面临何种大数据技术挑战，其最终都指向一个目的：相关活动的有效决策。大数据应用就是决策支持，即给定一个决策需求，通过获取数据和分析数据，最终形成决策依据，这与关系数据应用于联机事务处理、对象数据应用于多媒体数据管理和 XML 数据用于半结构化网络数据的整合交互等有着较大差异。

1. 大数据应用就是决策支持

从古到今，无论在政府行政管理、战场战争发展态势、商业竞争逐利演进、科学研究进展深化和日常工作生活导向等过程，成功和取胜的基础要素就是掌握更多更有价值的数据信息、能够在同一进程中做到比其他人知道更多、及时形成行动决心和比其他人更快更正确地实施决策步骤。

决策可以发生在任何情况下和任何场合中。从国家的总体宏观决策、战略格局部署和重大工程项目确定到企业部门的运营模式销售策略，再到个人周末的合适餐馆选择和度假的精准行车路线确定等。尽管决策的层面不同，但所倚仗的数据通常都比较巨量，具有多样的来源、多种的类型及时效性的限定等，这就增加了决策的复杂性和困难性。这里所涉及的数据应用通常具有跨界来源和跨界应用的特点，突破了原有领域行业界限，还会由数据的“量增”导致的人们决策方式的“质变”。这就是大数据应用语境中的决策问题。

在信息时代之前只能采用基于人工的决策方式，其特点是依靠手工收集数据，使用人脑积累的经验教训分析数据，然后凭借决策者自身的思维模式直觉地做出决策。

进入信息时代人们越来越多地使用借助计算机系统进行决策支持，如先期的计算机决策支持系统(Decision Support System，DSS)，而后的商业智能(Business Intelligence，BI)等，其特点是主要使用信息化手段采集和存储所需要数据，设计出相应计算机应用系统对所具有的数据进行样本分析，再与已有的成功样例进行比对校正，最终形成相应的决策。这里

的关键点是需要充分有效利用自身信息化积累的数据来开展决策。然而，自身的数据积累是一个漫长、费用高和困难的工作，一般只有政府部门和大型企业才会有实力如此实施决策。

进入 21 世纪后，随之而来的是互联网时代，由于计算机科学技术的不断进步和互联网设备的普及应用，不论是政府、组织、企业还是普通个人都越来越有能力获得决策所需要的各种数据。网络空间数据界中的这些数据来源不同并且类型多样，其数量和类型都可以超过原来政府部门和大型企业自身早起积累起来的数据。同时，由于数据存储和数据分析技术也取得了长足进步，涉及各类大数据的用户都有可能通过存储和分析所拥有的这些数据以期获得所需求的决策依据。由此，数据管理领域就出现了一种新型的决策方式，这种不同于常规数据决策支持的大数据应用具有维克托·迈尔-舍恩伯格及肯尼斯·库克耶在《大数据时代》一书指出 3 个特征，即由样本分析到全员分析、由精确分析到容错分析和由因果分析到关联分析。

2. 样本分析转向全员分析

无论是日常生活，还是在专业层面上用高级算法进行量化研究(如数据库查询统计等)人类活动都与数据有关。记录和存储工具的限制，先前人们往往只能收集相对于少量的数据用于数据分析。同时由于数据分析技术和硬件水平的限制，无法对全部采集到的数据进行全员上的总体处理，随机采样也就成为提取分析的常规方法。尽管基于统计学原理和数据挖掘等技术使得随机采样在某些领域取得成功，但随机采样过程存在着人们通常意识到却又有意避开的缺陷。

(1) 忽视细节微观信息。对于一个数据网络而言，如何得到合适的数据范围界定以及界定之后又如何获取满足"最优采样"标准的数据通常都是一个不易解决的课题。在实际问题中，随机采样得到的"小"网络难以反映总体网络所具有特性的情形时有发生，同时还会失去对某些特定子类进一步研究的能力。例如，对于信息化战场而言，无论怎样的数据采用都不足以满足情况各异且瞬息万变的战场，此时对所采用数据进行全员整体分析就成为决策应用的实际需求。

(2) 采样过程缺乏延展性。采样分析通常都要事先设计需要解决的问题，并按严密步骤设计与实施相应的随机采样过程，无法适应在调查过程中临时提出的各种要求。当需解决事先未考虑到的问题时就须重新采样。也就是说，采样分析缺少应用过程需要的延展性质。

收集指定的数据并通过随机采样对其进行分析，是在不可能完整收集全体数据和缺乏分析全部数据相应技术情况下的"无奈"选择，是信息处理能力受限的"技术性"产物。在大数据技术框架下，实际上具有对数据进行"全员样本"处理的应用需求和技术可能，通过掌握更为先进高效的数据分析技术使用所具有的全部数据以发现由于随机采用而被"人为忽略"而消失掉的数据，实现更高层面上的决策应用。

例如，人们主要是通过观察异常情况对电信和信用卡诈骗进行识别，但这只有掌握了所有相关数据才能做到。美国 Xoom 公司在处理跨境汇款业务过程中分析每一笔交易的所有相关数据。在 2011 年，Xoom 公司注意到用"发现卡"从新泽西州汇款的交易量比正常情况更多一些，于是就启动系统报警。如果只是进行随机采样，有可能用于分析的每笔或大部分交易数据都是合法的，但经过全样本分析，能够"一个不漏"地发现是否有犯罪集团在试图

诈骗。

3. 精确分析转向容错分析

从某种意义上来看，采用随机采样进行数据分析看作是"小数据时代"产物。在小数据时代，收集到的数据相对较少，而且受相应技术限制又只对其进行样本分析，同时还需确保分析过程中的数据尽量达到结构化和精确化以便精确计算以获取结论。也就是说，精确分析的思维方式实际上贯穿在采集、存储和分析的全过程中。

由"小数据"到"大数据"应用的重要转变之一就是必须允许由于存在不精确数据而对常规精确分析思维造成的挑战。对于大数据应用而言，当拥有巨量实时数据时，绝对的精准就不再是追求的主要目标，适当忽略微观层面上的精确度，允许一定程度的混杂与错误，反而可能在宏观层面上拥有更好的洞察预见力。在实际应用中，不精确有下述两种情形。

1）数据不精确

随着社交网络、电子商务和移动通信设备的普及及物联网和通信等技术的发展，通过简单廉价方法收集全员和全方位数据已成为可能，但也必须为此付出相应的代价，也就是说，某些错误的和不完整的数据也有可能进入到所收集的数据集合中。小数据时代绝不允许这样的事情发生，因为收集数据的有限性就意味着任何细微的错误都可能导致样本分析结果产生巨大偏差。在大多数情况下，人们都在致力于优化测量工具以取得更加精确的数据，如确定天体的位置、观测显微镜下物体的形状大小等。然而，大数据环境中收集的数据如此之多，无法对采集到的数据进行逐一的精挑细选和准确辨识，允许不精确的甚至有瑕疵的数据出现在大数据处理过程中已经成为一个新的常态而非必须加以克服的缺点。实际上，只有在放松容错标准前提之下，才能够采集、存储和分析各类大量的半结构化和非结构化数据。

2）结构不精确

不精确数据还具有数据结构模式方面的不精确性。人们研究发现，在实际应用过程中，人们采集到的大量数据只有5%的数据是具有精确结构而能适用于常规数据库存储管理，而95%的部分都是半结构化、非结构化或难以确定结构的数据。只有放宽对于数据结构和格式方面的精准性和一致性要求，才有可能达到所需要的庞大的数据规模要求，才有可能无须时刻关注某些数据出错而对整个分析的不利影响。这里，重要的关注点是从这些纷繁而不精准的大数据中获得裨益，而不是一味追求以高昂的代价去统一数据结构和类型以及消除所有的不确定性。

4. 因果分析转向关联分析

大数据应用方式还需要从因果分析思维转向关联分析思维，不再局限于千百年来形成的思维模式和固有偏见，以便更充分有效地获取和分享大数据带来的深刻洞见。

1）信息匮乏引致因果思维

通常人们多凭借直观意义明确的因果关系以寻求对自身所处世界的理解，执着于所面对现象背后的"前因后果"，并试图通过有限样本数据来剖析其中的内在机理。求证因果关系需要先假设一个因果关系的存在，然后再收集数据进行测试。但因果关系常常是难以证明的，因为从哲学上考虑，完全证实因果关系几乎不可能，而只能考虑某两者之间很有可能存在因果关系。事实上，由于所掌握的信息匮乏，没有更多的数据来解释说明某种现象，难以通过有限样本数据揭示事物之间普遍的关联关系，只能转向在有限的数据中寻求因果关系，因此形成了借助因果关系解释说明问题的常态化分析思维模式。由于长期以来人们多

以习惯了信息的匮乏，进而也就习惯了在少量数据的基础上进行推理思考和论证因果关系。

在大数据环境中，由于拥有如此之多的数据和更为有效的数据分析手段工具，可以通过大数据技术更快更容易地挖掘出事物之间隐蔽的关联关系，获得更多的认知与洞见，并运用这些认知与洞见帮助人们去捕捉当前和预测未来。

2）注重“其然”而非“所以然”

关联思维的关键点在于量化两个数据值之间的数理关系。关联关系的“强”是指当一个数据值增加时，另一个数据值很有可能也会随之增加。例如，谷歌就研究了当流感疫情暴发时，在一个特定的地理位置，越多的人通过谷歌搜索有关疾病的特定词条，实际上该地区就有更多人患上流感。这两者之间表现出“强”的关联关系。相反，关联关系的“弱”就意味着当一个数据值增加时，另一个数据值出现的变化趋势状态和该数据及前一数据的关联情形。

关联关系通过识别有用的关联物来分析一个特定现象，而不是试图去揭示其内部的运作机理。对于关联关系而言，只有可能性而无绝对性。例如，电子商务网站通过不同商品销量提取关联性，为顾客推荐与其购买过的商品关联关系“强”的商品，但并不表示网站推荐的每个商品都是顾客想买的商品，只能说该商品被一起购买的可能性高。实际上，管理者并不需要知道购买A商品的顾客“为什么”会同时购买B商品，只需知道“他购买了B商品”就足够了。在很多情况下，这种“知其然”而不必非要“知其所以然”就足以产生相当可观的价值，那么按照奥卡姆剃刀原则，自然就没有了因果分析的逐利驱动。在大数据应用过程中，人们并不需要非得让自己绞尽脑汁去探究现象背后的原因，完全可以让数据自己“发声”，以便使用关联关系做到比过往更容易、更快捷和更清楚地分析事物。这种运用关联思维的洞察力足以重塑很多行业，开辟出全新的天地。

在大数据时代来之前，关联关系已被证明其用途。由于可供使用的数据较少特别是由于收集数据费时费力，统计学家们也会证明一种关系时寻找另一个关联物，然后再收集与之关联的数据进行关联分析。同时还会使用建立在理论基础上的某些假设来证明这个关联物是否真的合适，这样通过多次反复尝试才可得出合适或不合适的结论。在这个相当烦琐的过程实际上是一种人工选择过程，个人或团体的偏见很容易导致在设立假设、应用假设和选择关联物的过程中出现偏差。对于大数据而言，通过建立在人的主观断测基础上的关联物监测方法已经难以为继，因为数据量太大需要考虑的领域太过复杂，无法进行关联物的人工选择。而且在强大机器计算能力环境中，不再需要只采样一小部分相似数据而可依据全部数据进行关联分析。

3）关联思维预测未来

以关联关系分析为基础进行预测是大数据应用的核心要素之一。如果事件A和事件B经常一起发生，就可在关注到事件B发生情况下对事件A发生甚至和A一起发生的其他事件进行预测。这种关联预测分析方法已经被广泛地应用于相关领域。安大略理工大学的卡罗琳·麦格雷戈(Carolyn McGregor)用软件监测处理即时的病人信息。在早产儿的病情诊断系统中，系统会监控16个不同的数据，如心率、呼吸、体温、血压和血氧含量，这些数据可以达到每秒钟1260个数据点之多。在早产儿出现明显感染症状的24小时之前，系统就能监测到早产儿细微的身体变化发出的感染信号。这些信号人类无法用肉眼看到，但通过计算机却可以得到。这个系统依赖于关联关系而非因果关系，从而使得麦格雷戈发现了一些与医生的常规见解相悖的关联关系，如某些早产儿出现稳定的生命体征却可以表明病人

发生了严重的感染，而一般认为只有恶化的生命体征才是全面感染的征兆，但这可能就是直觉犯下的错误。对于这种在巨量数据基础上找出的隐含关联性，人们会猜测早产儿的“稳定”的生命体征可能并不是病情好转的标志，反而表明器官是在做好抵抗病情进一步恶化前的必要准备。实际上也许谁也无法证明这是不是原因所在，但由于麦格雷戈发现了这种关联关系，在实际中就挽救了不少早产儿的生命。

4）更多类型的关联分析

如前所述，在小数据时代人们也有实行数据应用的关联分析，但由于计算能力的不足和所用技术的限制，大部分都局限于寻求获取线性关联关系。随着大数据时代的到来，充分大的数据量和更为有效的数据分析技术使得人们能够发现隐含于大数据中更为复杂的“非线性关联关系”。例如，通过大数据分析，社会上的收入水平和幸福感实际上会呈现出一种非线性关系，而这在小数据时代一直被认为是线性正比关系。由线性进入到非线性是人们分析思维进入到新的更高层级的标志。如此得到这样的非线性关联关系对于决策者拟定如何提升全民幸福感方案非常重要。在大数据环境中，新的分析工具和思路为人们提供了一系列新的视野和有用的预测，使得人们看到了很多以前不曾注意到的实体之间更为精细和深入的内在关系，通过探求“是什么”而不在执着寻找“为什么”，正是人们分析思维模式的一次有意义的扩展，有助于掌握以前无法理解的事物变化发展动态。

9.1.4 大数据理念认知

由上述讨论可知，作为数据集合，大数据来源于网络空间中的数据界，具有 PB 级容量、数据产生到数据消耗时间窗口严格限定等数据自身特征；作为数据管理技术，具有由其数据特点确定的 4V 技术特征；大数据应用驱动可以由其产生来源予以说明，因此，从领域应用角度来看，大数据应用就是决策应用，决策建立在数据分析基础之上，从而就具有由样本分析到总体分析、由精确分析到容错分析和由因果分析到关联分析的大数据应用特征。

大数据概念可以是一个伞形概念，但不是可以任意滥用的定义，否则就会出现“新瓶装旧酒”的影响，似乎大数据只不过是现有数据管理技术的另一种包装或整合。

上述讨论实际上说明，大数据概念应该具有自身严格的逻辑内涵和明确的外延界定，只不过由于涉及因素众多，呈现出较为复杂的情形。

1. 大数据概念

针对时常出现的对大数据概念的混用和滥用情形，人们在基于前述讨论基础上，对大数据进行逻辑层面上定级分解描述。

大数据所涉及的主要有下述三类人群。

（1）大数据拥有者：他们主要关注大数据的自身特质和描述。在其观念中，大数据就是一种具有前述讨论自身特征的特定的数据集合。

（2）大数据研究者：主要是数据科学领域中的学者和研发人员，他们主要关注于大数据的科学属性和管理技术，在其理念中，大数据是具有 4V 特征的一种计算机理论与技术。

（3）大数据使用者：主要是计算机领域外的各类人员，他们主要关注于大数据的应用也就是决策支持，在其理念中，大数据就是用以支持决策活动，是一种新的和有效的决策方法。

之所以对于大数据概念会缺少明确的概念辨识与理解认知，可能就是没有仔细分清上

述3个层面上的人们对大数据理念认知上的差异。

从逻辑上考虑，在厘清理念层级认知基础上，人们或许将大数据概念表述为“向决策支持提供服务的大数据集合、大数据技术和大数据应用总称”更为合适，实际上是将大数据概念置于比“大数据集合”“大数据技术”和“大数据应用”理念层次更高的逻辑层面，有利于分清级别，理顺脉络，也有助于人们对于大数据概念的深入探讨(具体可见本章后面所列的参考文献)。

2. 大数据意义

在大数据时代，将会有更多种类的预测、更为准确的预判、更加缺少的隐私，而这就在可能标志着人们对社会交互、经济活动和日常生活等方面的全新认知。实际上，大数据也许会让人类对客观事物的认识回归本源，同时，大数据通过影响经济生活、政治博弈、社会管理、文化教育、科研、医疗保健和休闲娱乐等行业，与每个人都会产生密切关联。人们需要正视这股浪潮，积极顺应这股浪潮，了解和认识大数据带来的深远而广泛的意义。

1) 数据成为资源和财富

网络时代，信息为本。庞大的数据存储具有常规数据所难以比拟的潜在价值，而大数据的核心就是挖掘出这种独有价值。只要能够提取出其中隐藏的关联信息，看似纷乱庞杂的各型数据就会变为有用珍宝。

(1) 数据是基础性资源：数据已成为可与土地、资金、物质资产和人力资本相提并论的重要基础性资源，用以提高企业和公共部门的生产率和竞争力，并大幅度提高消费者福利。2011年麦肯锡研究院数据显示，制造业生产商利用大数据可以使产品研发、组装成本减少50%，运营成本减少7%；掌握全球个人位置信息，每年可获得6000亿美元的消费价值。

(2) 数据是战略性资源：人类文明经历农业经济和工业经济之后，将进入互联网经济这种新的社会经济发展形态，此时，数据将成为企业、社会和国家层面重要的战略资源。欧美等发达国家把数据资产上升到国家信息战略高度，欧洲发达国家政府行政管理利用大数据，节省至少1000亿欧元的成本；美国医疗行业每年通过大数据获得的潜在价值可超过3000亿美元，能够使得美国医疗卫生支出降低超过8%。

(3) 数据是重要生产力：随着信息网络技术在国民经济各领域的不断渗透和扩散，各产业部门界限被打破，人们预测未来最大的能源可能就是大数据的数据收集、拥有和汇总能力。掌握数据挖掘、分析和利用的技术，就是掌握了互联网经济时代重要的先进生产力。

2) 新技术革命的助力推动

大数据带来的不仅是思维的变革，还有新技术与制度的变革。

基于社会化网络的平台和应用使得数以百亿计的机器、企业和个人随时随地获取和产生新的数据，由此正在引发全球范围内深刻的技术变革。既有的技术架构和路线，已经无法高效处理如此巨量的数据。

(1) 大数据具有数据量大、数据类型多的特征，这使得常规数据库设计面临挑战。常规的数据库引擎要求数据按事先设计好的形式高度准确排列，数据被整齐地划分为包含“域”的记录，每个域都规定了特定种类和特定长度。关系数据库主要是基于“数据稀缺”而设计开发的，因而能够也必须做到仔细规划，使得存储数据显现规律，便于统计分析。现在，人们面临的是各种各样和参差不齐的巨量数据，无法预先设定数据种类和存储长度，常规数据存

储和分析方法与之出现了明显冲突。这是对现有数据库技术的尖锐挑战，但也必将推动新的数据库原理技术的产生发展。

(2) 面对大数据越来越实时的要求，不仅是数据流动快，还是对大数据分析、处理速度的要求，云计算也就成为大数据管理系统技术实现过程中不可忽视的重要组成部分。

(3) 大数据可以全方位囊括人-人互联、人-物互联和物-物互联过程中的所有数据，其中人-物互联和物-物互联产生的数据量在理论上应该大大超过人-人互联过程的数据量。人-物和物-物互联的技术过程就是人们常说的“物联网”。物联网相当于整个互联网的感觉和运动神经系统，通过收集人类日常网络活动中留下的痕迹获取大量数据。物联网实际上也是(广义)大数据框架下的重要组成部分。

(4) 随着互联网、移动互联网快速发展和智能信息终端设备的广泛普及，每个人几乎每时每刻都在产生数据。由于在公用网络上运行，所产生的数据从本质上来说将不再神秘和具有私有特性。人们的相关数据可以在毫无知晓的状态下被有关机构和企业收集、存储和使用。当数据形成资源而体现巨大价值时，数据外泄也许是所有人不可避免的困扰，而大家所面临的共同问题是如何保护自己的隐私权。信息安全问题早已有之，大数据只是将这一问题以更加鲜明的色彩和更为迫切的驱动展现在人们面前，也就是到了常说的“非解决不可”的地步。可以相信，由于大数据已经融入社会生活的各个方面，解决人们长久关注而又未能有效解决的数据信息安全问题一定能够得到更加强有力的推进与发展。

3) 社会发展的机遇促进

大数据对社会发展产生的影响并不限于技术层面，它从各方面促进了社会的发展。

(1) 影响经济活动和社会生活。大数据解决方案消除了常规情况下数据计算和数据存储的局限。借助于不断增长的私密和公开数据，为大数据客户带来新的实质性的收入增长点及富于竞争力的优势，一些新的商业和经济活动模式正在兴起，促进了社会进步发展，人们的一些社会生活习惯也随之改变。

① 商业模式方面：大数据正在重构很多常规行业。通过收集、整理生活中方方面面的数据，并对其进行分析挖掘，进而从中获得有价值的信息，最终衍化出新的商业模式。大数据的应用越来越彰显它的优势，它占领的领域也越来越大，如电子商务、O2O、物流配送等，各种利用大数据进行发展的领域正在协助企业不断地发展新业务，创新运营模式。

② 企业管理方面：大数据彻底改变了企业内部运作模式。以往管理是“经理如何讲”，现在正逐步变成“大数据怎么说”，这就是对常规领导力的挑战，也是对企业管理岗位人才新的界定，即不仅需要懂得企业业务流程，还要成为数据专家。跨专业的要求改变过去领导力主要体现在经验积累和过往业绩，更需要熟练掌握大数据分析工具，善于运用大数据分析结果进行企业的销售和运营管理。

③ 医疗卫生方面：突出表现在大数据提供了个性化的医疗服务。在大数据帮助下，医生在诊疗时通过对一个患者的累计历史数据进行梳理，综合各种案例分析，并结合遗传变异、对特定疾病易感性和对特殊药物反应性等关系，实现个性化的医疗。还可以在患者发生疾病症状前，提供早期的检测和诊断。

④ 社会安全管理方面：实时掌握动态的流动人口来源和出行，实时交通客流信息及拥堵情况。利用短信、微博、微信和搜索引擎等，收集热点事件、挖掘舆情和追踪谣言信

息源头。

⑤ 劳动就业方面：据盖特纳咨询公司预测，大数据将为全球带来440万个IT新岗位和上千万个非IT岗位。麦肯锡公司预测美国到2018年需要深度数据分析人才为44万～49万人，缺口为14万～19万人；需要既熟悉本单位需求又了解大数据技术与应用的管理者150万，这方面的人才缺口更大。中国人口世界第一，不仅拥有巨大海量的大数据，而且是人才大国，但能理解与应用大数据的创新人才现在还是稀缺资源。

(2) 科学研究突破。随着大数据的快速发展，就像计算机和互联网一样，大数据很有可能引发新一轮的技术革命。随之兴起的数据挖掘、机器学习和人工智能等相关技术，可能会改变数据世界中现有的很多算法和基础理论，形成一门"数据科学"的新兴数据管理技术领域。近年来，美国哥伦比亚大学、纽约大学、加州大学、卡耐基-梅隆大学等许多高校建立数据科学研究机构或开设数据科学专业研究生培养项目。以大数据研究为基本特征的数据科学也会对其他前沿学科产生深远的影响。例如，欧洲核子研究组织通过对大约200PB的数据用150个计算中心进行长达3年的大数据分析计算，在2013年3月14日宣布了希格斯玻色子的确认，这是前沿科学领域中一项具有重大影响的成果。

作为一种计算机技术，大数据不仅对于相关领域的技术创新产生巨大驱动，同时还由于其特有的品质和功效，对于人们的社会生活及文化活动都产生了重要而深远的影响，这正是大数据技术的意义所在。

9.2 大数据基本技术

如前所述，大数据的主要内涵是从体量庞大、类型众多的数据集合中获取有用的决策信息，因此大数据基本技术就是实现大数据价值的关键，而这就需要灵活运用多学科的原理与方法。目前，来自于统计学、计算机科学、应用数学和经济学等领域的技术已经应用于大数据整合、处理、分析和可视化；同时，已有的一些面向规模较小、种类较少的数据开发技术也被重新成功地应用于更多元的大规模数据集。从技术上看，物联网、移动互联网和常规互联网，每天都源源不断地产生海量数据，大部分是半结构化数据和非结构化数据。这些数据存储到关系型数据库用于分析将会代价过大，如Google在使用传统手段处理网页文件的倒排索引时就曾经花费了数月时间，这种情形催生了各类相关的新型数据库技术产生与发展。同时，大数据需要对数据进行分析、提取信息，无法仅使用单台计算机进行存储和处理，必须采用一种新型的分布式架构，所以大数据技术也和云计算紧密相连。大数据处理技术基本流程如图9-2所示。

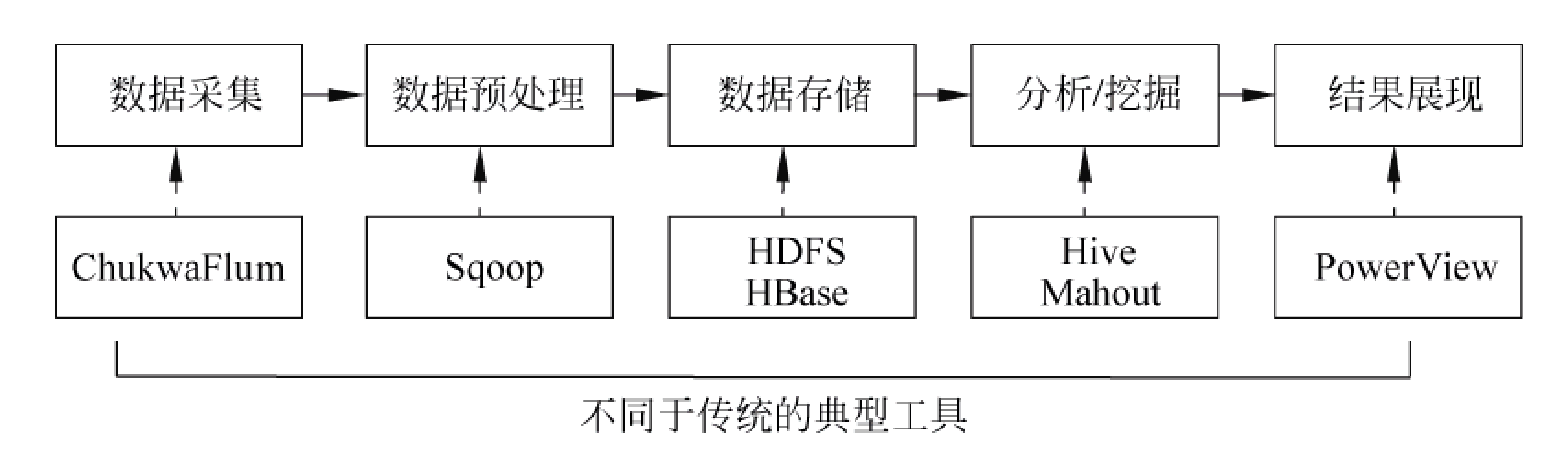


图9-2　大数据处理技术基本流程

本节将简要介绍大数据框架下的数据采集与预处理、数据分析与数据挖掘即呈现等通用技术、大数据与物联网及云计算关系等相关知识。

9.2.1 数据采集

数据采集是大数据挖掘和分析的重要基础，有效的数据采集对大数据挖掘研究具有十分重要的意义。

大数据采集可以具有多种多样的渠道。在智能交通中，有基于GPS定位信息采集、基于交通摄像头的视频采集和基于交通卡口的图像采集等；在互联网上，主要是针对网络媒介（如搜索引擎、新闻网站、论坛、微博、博客、电商网站等）的各种页面信息和用户访问信息的数据采集，采集内容包括文本信息、URL、访问日志、日期和图片等；在卫生保健领域，数据采集渠道除了常规信息系统平台外，还可通过移动APP、智能终端、大型医疗设备、健康监测设备、基于测序仪和可穿戴设备等多种方式进行采集。

针对不同的数据采集渠道，需要采用不同的数据采集方法，主要有下述技术方式。

（1）互联网企业数据采集：自身大数据采集多采用系统日志方式，如Hadoop的Chukwa，Cloudera的Flume，Facebook的Scribe等。这些方式采用分布式架构，能满足每秒数百MB的日志数据采集和传输需求。

（2）一般网络数据采集：通常通过网络爬虫或网站公开API等方式从网站上采集获取数据信息，将非结构化数据从网页中抽取出来统一存储为本地数据文件，存储方式多采用结构化类型。此种方式支持图片、音频、视频等文件或附件的采集，附件与正文可以自动关联。

（3）网络数据流量采集：通常除需要采集网络包含数据外，还可能需要采集网络数据流量，可以使用DPI或DFI等带宽管理技术进行处理。

（4）保密性数据采集：对于企业生产经营数据或学科研究数据等保密性要求较高的数据，通常通过与企业或研究机构合作，使用特定系统接口等相关方式采集数据。

9.2.2 数据预处理

现实世界中的数据大多不够完整或不尽一致，无法直接进行数据分析或分析挖掘结果不甚理想。数据预处理就是对采集到的数据进行填补、平滑、合并、规格化和检查一致性等处理，并对数据的多种属性进行初步组织，提高数据分析质量，减少分析挖掘时间，这对于体量巨大的大数据来说尤为重要。如果不对数据进行事先预处理而只是单纯依赖服务器的计算能力，通常难以满足大数据对处理速度和处理精确性等要求。

与常规数据预处理流程相似，大数据预处理通常是指ETL，基本步骤也是将多个数据源中的数据抽取到临时中间层后进行清洗、转换和集成，最后加载到目标数据库或相应文件存储系统中作为数据分析的基础。

（1）数据抽取。负责将各个数据源中的数据提取出来，其中包括全量抽取和增量抽取。全量抽取是指将数据源中的数据从数据库中原样抽取出来，增量抽取是指只抽取截止到上次抽取时间结点后数据库中新增或修改的数据。目前增量抽取应用更为广泛。对于获取增量，首先要求是准确性，能够将业务系统保护的数据按照一定频率准确地捕获；其次要求是良好性能，捕获变化数据的过程，不能对业务系统造成太大的压力，不能影响业务系统的正常运行。

(2) 数据转换。从数据源抽取的数据不一定完全满足目标数据库的需求，如数据格式不一致、数据输入错误和数据不完整等，因此需要对抽取的数据进行转换。数据转换一般包括数据过滤、数据替换、字段映射、数据清洗、数据计算、数据验证、数据加解密、数据合并和数据拆分等步骤。数据预处理中，花费时间最长的就是数据的转换部分，一般情况下这部分的工作量要占整个数据预处理的2/3。

(3) 数据加载。将转换和汇总的数据加载到目标数据库或相应文件存储系统中，并可实现批量加载。数据加载所采用的技术方法由数据操作类型和数据体量来决定，一般可以通过SQL语句的方式，也可以采用批量装载的方式。转载步骤中的关键组件是代理键管道，代理键管道主要用于将加载完成的数据表中的自然键替换为代理键，在代理键管道内，维度表的主键与外键仍然得到保留；但是为了提升系统性能，在完成加载以后，一些约束条件将被去除而仅保留自然键进行。

大数据预处理方式具有以下基本特点。

① 多维数据处理：由于大数据中存在文本、图像和视频等多种非结构化数据，这些非结构化数据可以从不同角度描述，常规数据描述方法不能满足大数据的多样性需求。在预处理过程中，需要寻找这些数据的不同属性，从多个维度对数据进行描述，提升数据可解释性，灵活应对分析需求的变化。

② 大规模并行处理框架应用：使用MapReduce等并行处理方式来进行数据平滑和聚类等预处理工作。

③ 分布式存储系统及数据流式处理：大数据导入的数据量巨大，可达到每秒GB级的量级，需要在预处理之后，将其导入到分布式存储系统中。对支持实时流处理的数据，则可先进行实时流式的内存简单预处理，在把流式处理结果存储到分布式系统以供后续分析使用。

9.2.3 数据存储

随着大数据时代的到来，需要存储的数据越来越多，数据也呈现越来越复杂的结构。如何对海量数据进行组织与存储就成为了一个基本点。具体要求就是对数据流在加工过程中产生的临时文件或加工过程中需要查找的信息进行有效组织，并以某种格式记录在计算机内部或外部存储介质上。

1. HDFS分布式文件系统

作为分布式计算的存储基础，HDFS是Hadoop框架的分布式文件系统，它负责数据的分布式存储及数据的管理，并能提供高吞吐量的数据访问。

HDFS体系框架是经典的Master/Slave结构，一个典型的HDFS通常由单个NameNode和多个DataNode组成。NameNode是一个中心服务器，负责文件系统的命名空间的操作。DataNode一般是一个结点部署一个，负责管理它所在结点上的存储。从内部来看，一个文件其实被分为一个或多个数据块，这些块存储在一组DataNode上，负责处理文件系统客户端的读/写请求，在统一调度下进行数据块的创建、删除和复制，如图9-3所示。

HDFS针对数据读/写的应用场景，具有“一次写、多次读”的特征；数据写操作主要是顺序写，即在文件创建时的写入或现有文件之后的添加操作。HDFS保证一个文件在一个时刻只被一个调用者执行写操作，但可被多个调用者执行读操作，如图9-4和图9-5所示。

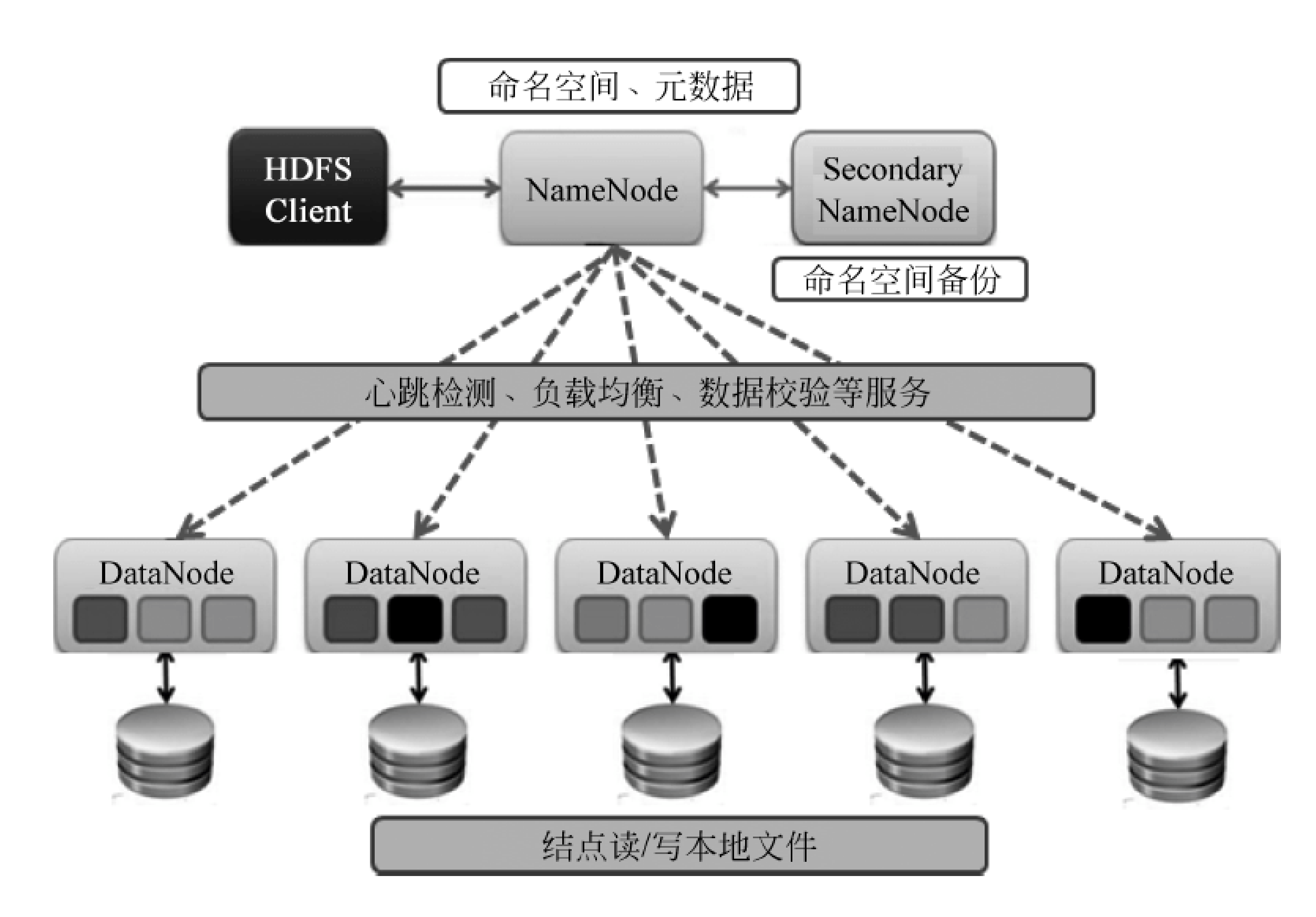


图 9-3　HDFS 文件的存储结构

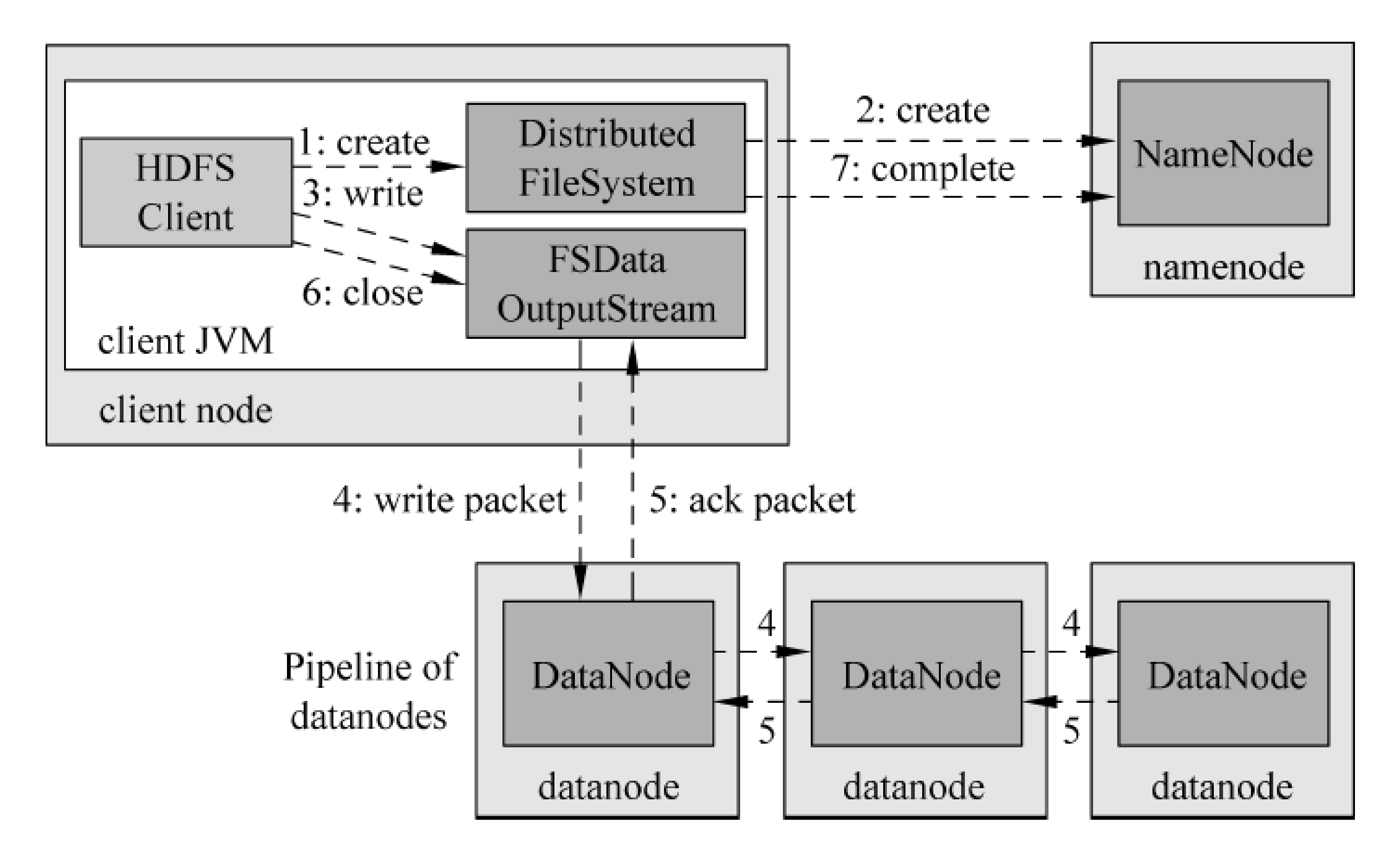


图 9-4　HDFS 写操作

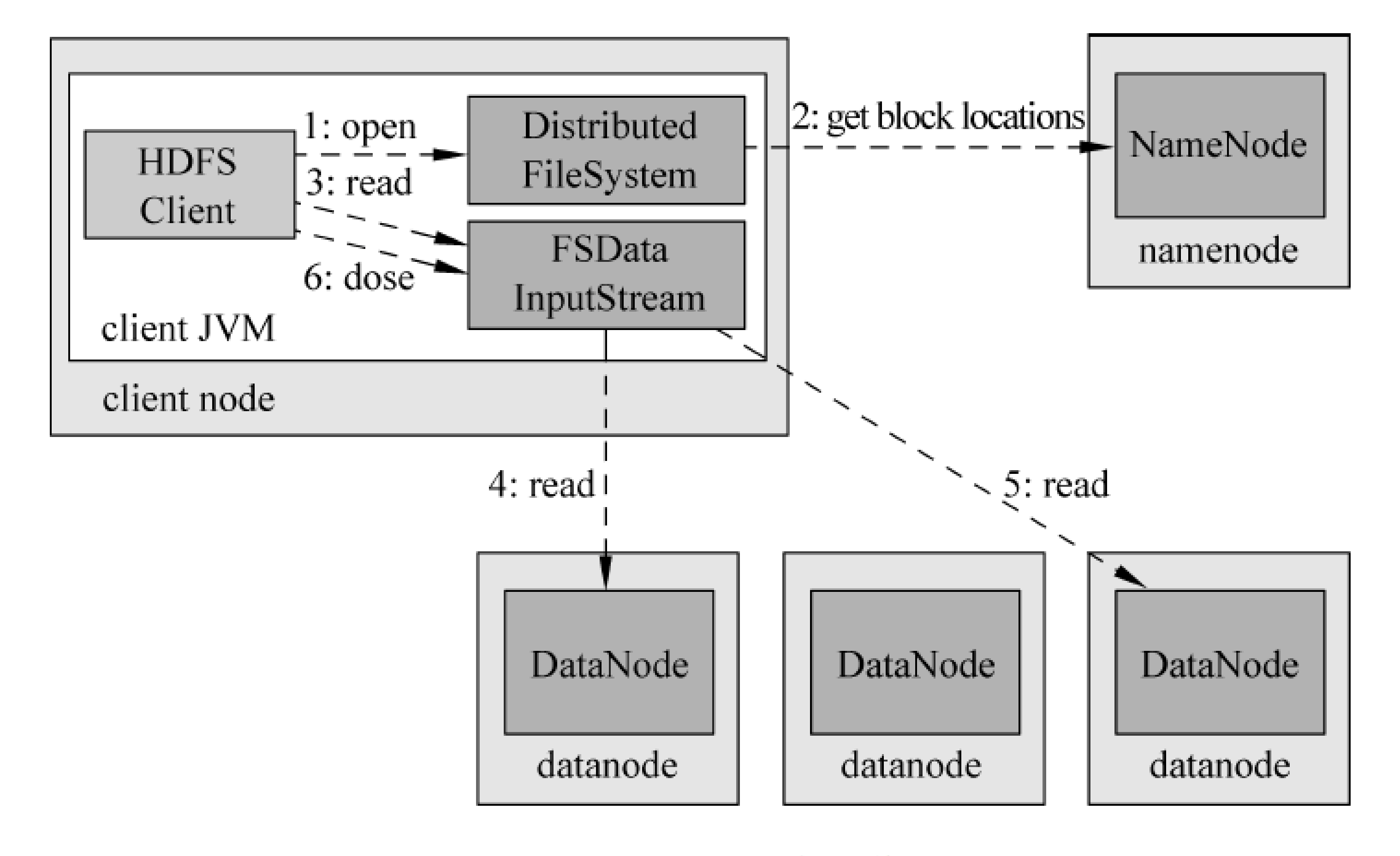


图 9-5　HDFS 读操作

HDFS 采用“块”作为其管理的文件存储单元。块通常为 64MB 或 128MB。一个块数据位于一个结点上，而一个大型文件可能存储在多个块的单元中，一个文件所在块可以位于不同的结点。

HDFS 中的命名结点以维护文件系统的逻辑结构，文件的元信息包括文件数据实际存储位置信息，它是文件系统可访问的关键。命名结点将文件系统的信息存储在内存中以供快速访问和操作，同时将数据持久化存储到本地文件系统和远程文件系统中以备故障恢复。

HDFS 基本特点如下。

(1) 高容错性：数据自动保存多个副本，副本丢失后能够自动恢复。

(2) 适合批处理：移动计算而非数据，数据位置暴露给计算框架。

(3) 适合大数据处理：GB、TB 或 PB 级数据，百万规模以上的文件数量，10K＋结点规模。

(4) 流式文件访问：一次性写入，多次读取，保证数据一致性。

(5) 可构建在廉价机器上：通过多副本提高可靠性，提供了容错和恢复机制。

2. NoSQL 非关系型分布式数据库

使用计算机进行数据管理就是应用数据库平台存储和处理数据，将大数据作为“数据”进行管理，就需要建立相应的数据库平台，基于大数据环境的新型数据库技术也就成为大数据相关技术中的重要基础之一。大数据是一种包含结构化、半结构化及非结构化的“混成”数据，不能只是对 RDB 进行“设备”层面上“量”的扩充，而需要进行数据结构体系层面的拓展与补充。

大数据思维从精确思维转向容错思维，即允许“不精确”数据的存在。RDB 引擎要求数据高度精确且准确排列，数据库中数据文件都具有二维关系表的固定结构格式，每个元组字段的结构组成需要一致，即使不是其中元组需要的字段，数据库也会为其相应字段分配数据值，如分配空值。这种结构虽然便于进行关系表之间的连接等数据操作，但对于大数据中常见的关联查询却效率低下，也难以满足大数据的可扩展需求。

大数据拥有各种各样、参差不齐的巨量数据，而且大数据所要解决的问题可能只有在数据收集和处理过程中才会被发现与提出，关系数据存储和分析的方法也无法适应大数据对数据库的高并发读/写、海量数据存储和复杂的关联分析和挖掘需求。

正是上述应用需求催生和驱动了面向大数据的新型数据库技术的研发，这些技术扩展了常规 RDB 数据规范性和结构规范性，形成了近年来出现的各类非关系型数据库系统。由于这些数据能够处理超大量半结构和非结构化数据，弱化了数据库的关系型特性，简化了数据库中的数据组织，便于对数据和系统架构进行扩展，是一种“非 SQL”数据库系统，因此通常称为 NoSQL 数据库，其特点在于可以自定义数据存储格式。大数据通常采用分布式存储，非关系型分布式数据库(NoSQL)是分布式存储的主要技术。

NoSQL 概念最初见于 2009 年，常见语义解释可以是 Non-Relational。由于在通常情况下，实际上人们已将关系数据查询语言 SQL 作为关系型数据库代名词，Non-Relational 也就相当于 NoSQL，进而 NoSQL 也被解释称为了 Not Only SQL，泛指各类新出现的非关系型数据库。

NoSQL 数据存储不需要固定的表结构，因此通常没有数据的连接操作，各个数据可以独立设计，数据结构也不是固定的，这就易于将数据分散到多个服务器中，减少了单个服务

器的数据量，能够以更好的相对于关系数据库的性能优势处理超大量的数据。

相比起常规关系型数据库，NoSQL 具有以下几方面的基本特征。

（1）易扩展性。NoSQL 数据库共同特点是去掉 RDB 的关系表结构特性，数据之间可以没有结构一致性方面的关联，容易扩展，由此也导致了架构层面上可扩展能力的增强。

（2）大数据量高性能。由于数据之间的无关联性，数据库结构相对简单，在大数据量环境中具有更高的读/写性能。

（3）灵活的数据格式。无须事先设计和建立数据字段，随时可以存储自定义的数据格式，使得数据库的维护更加方便灵活。

（4）高可用性。在对性能影响不大的情况下就可以方便地实现高可用的架构。

目前主要有 4 种非关系型数据库管理系统，即基于 Key-Value 键值存储、基于列存储、基于文档的数据库和图形数据库。

1）键值(Key-Value)存储数据库

Key-Value 数据库以键-值的形式存储数据。根据数据的保存方式可以分为临时性、永久性和两者兼具 3 种。

（1）临时性键-值数据库：将所有数据存储在内存中，保存和读取的速度更快，但当数据库停止工作时，数据就不复存在，或者当数据超出内存容量时，旧数据也会丢失，临时性也就包含有“数据有可能丢失”的意思。例如，Memcached 数据库就是临时性的。

（2）永久性键-值数据库：将数据保存在硬盘中，数据不会丢失，此即“永久性”的含义。由于对硬盘的 I/O 操作，与临时性数据库相比具有性能上的差距，但其最大的优点是不会丢失数据。例如，Tokyo Cabinet、Flare 等都属于该种类型。

（3）临时、永久兼具键值数据库：综合两者优点，如 Redis，是一个高性能的 Key-Value 数据库，首先把数据保存在内存中，在满足一定条件（如过多久或多少数据被保存之类）后，通过异步操作将数据写入硬盘。这样既兼顾了性能，又保证了永久性。

这一类数据库主要会使用一个哈希表，表中有一个特定的键和一个指针指向特定的数据。键-值模型对于 IT 系统来说的优势在于简单、易部署及能够提高读/写性能需求。

2）列存储数据库

列存储数据库改变了常规数据库以行为单位存储数据的模式，而以列为单位存储数据，这样做的好处是可对大量行的少数列进行读取，也可对所有行的特定列进行同时更新。此类数据库通常用来应对分布式存储的海量数据。例如，Cassandra 和 HBase 属于此类数据库。

Cassandra 由 Facebook 于 2008 年开源发布，随后 Facebook 自身也使用 Cassandra 的另一个不开源的分支。开源 Cassandra 主要由 Amazon 的 Dynamite 团队维护。目前除 Facebook 之外，twitter 和 digg.com 都在使用 Cassandra。

列存储数据库将数据按行排序，按列存储，将相同字段的数据作为一个列族来聚合存储。不同的列族对应数据的不同属性，这些属性可根据需求动态增加，通过这样的分布式实时列式数据库对数据统一进行存储和管理，满足高可扩展性的需求，避免了常规数据存储方式下的连接查询。当只需查询少数几个列族的数据时，采用列存储方案进行大数据存储可大大减少读取的数据量，减少数据装载和 I/O 的时间，提高数据处理效率。对于图像、视频、URL 和地理位置等类型多样的数据，采用这种面向列存储的数据库可更为有效地组织管理数据。

3）文档型数据库

文档型数据库基本特征是可以在不定义表结构环境中，也可像定义了表结构一样来使用相关数据，从不需要为保持数据的一致性而付出系统开销，同时，还可以通过复杂的查询条件获取数据。文档型数据库可看作是键-值存储数据库的升级版，但比键值数据库具有更高的查询效率，能够满足海量存储需求和访问。文档型数据库主要解决的不是高性能的并发读/写问题，而是要在保证海量数据存储的同时使得系统具有良好的查询性能。例如，CouchDB 和 MongoDB 等属于此类数据库。

4）图形(Graph)数据库

图形数据库使用灵活的图形模型并可扩展到多个服务器上。图形数据库没有标准的查询语言，因此进行数据库查询需要制定数据模型。许多图形数据库都有 REST 式的数据接口或查询 API。例如，Neo4J、InfoGrid 和 Infinite Graph 属于此类数据库。

上述各种类型的 NoSQL 数据库如表 9-1 所示。

表 9-1　NoSQL 数据库分类

分　类	举　例	典型应用场景	数据模型	优　点	缺　点
键-值（Key-Value）存储数据库	memcached、Tokyo Cabinet、Redis、Flare	内容缓存，处理大量数据的高访问负载，也用于一些日志系统等	Key 指向 Value 的键-值对，通常用 hashtable 来实现	查找速度快	数据无结构化
列存储数据库	Cassandra、HBase、Riak	分布式的文件系统	面向列存储	查找速度快，高扩展性，更易进行分布式扩展	功能相对局限
文档型数据库	CouchDB、MongoDB	Web 应用	不需事先定义表结构	数据结构要求不严格，可满足负责查询需求	查询性能不高，而且缺乏统一的查询语法
图形数据库	Neo4J、InfoGrid、Infinite Graph	社交网络，推荐系统等。专注于构建关系图谱	图结构	利用图结构相关算法，如最短路径寻址等	需要对整个图做计算才能得出需要的信息，不易做分布式的集群方案

3. 虚拟存储技术和云存储技术

为实现存储的低成本、高可扩展性和资源池化，需要用到虚拟存储技术和云存储技术。

1）虚拟存储技术

虚拟存储技术是指将存储系统的内部功能从应用程序、计算服务器、网络资源中进行抽象、隐藏和隔离，最终使其独立于应用程序、网络存储和数据管理。相比于传统的存储，虚拟存储技术磁盘利用率高，存储灵活，管理方便，并且性能更好。

2）云存储技术

云存储是由云计算概念延伸并衍生发展而来的一个新的概念。云存储凭借分布式文件系统、集群应用、网络技术等功能，通过应用软件将网络中大量不同类型的存储设备集合起来协同作用，实现对外共同提供数据存储及业务访问功能的一个系统。这样，既保证了数据的安全性，也节约了存储空间。

9.2.4 数据处理

1. MapReduce基于并行计算的分布式数据处理

Hadoop MapReduce是一种分布式海量数据处理框架。其采用主从结构，在一个MapReduce集群中有一个控制结点和多个工作结点。当集群运行时，所有的工作结点会定期地向控制结点发送心跳信息，报告本结点的当前状态。收到心跳信息后，控制结点会根据当前的工作情况和工作结点自身的状态给工作结点发送指令信息。控制结点根据收到的指令信息会完成相应的动作。MapReduce框架实现的是跨结点的通信，擅长横向扩充、负载平衡、失效恢复和一致性等功能，适合有很多批处理的大规模分布式应用，如Web索引建立等。

基于MapReduce写出来的应用程序能够运行在由普通机器组成的大型集群上，并以一种可靠容错的方式并行处理TB级以上的数据。这允许没有任何并行和分布式系统经验的编程者轻松利用一个大型分布式系统中的资源。在MapReduce框架中，用户进行的数据处理工作的基本单位是"作业"，"作业"被分为Map和Reduce两个阶段来执行，而每个任务在每个阶段又有多个任务在并行执行，这些任务被分配到多个工作结点上执行，完成基本的数据处理工作。其中，在Map阶段从分布式文件系统中读取数据，并且将输入数据转换为键-值对输出，经过shuffle过程，即划分、合并和排序之后，具有相同键的键-值对会被聚合在一起，交给Reduce任务处理。Reduce任务一次会读取所有键系统的键值对进行处理，处理后的结果会输出到分布式系统。在作业执行过程中，系统会自动完成作业和任务的监控、调度和容错，用户只需简单地实现相应的接口即可。

MapReduce也采用了Master/Slave(M/S)架构。它主要由以下几个组件组成：Client、JobTracker、TaskTracker和Task，具体如图9-6所示。

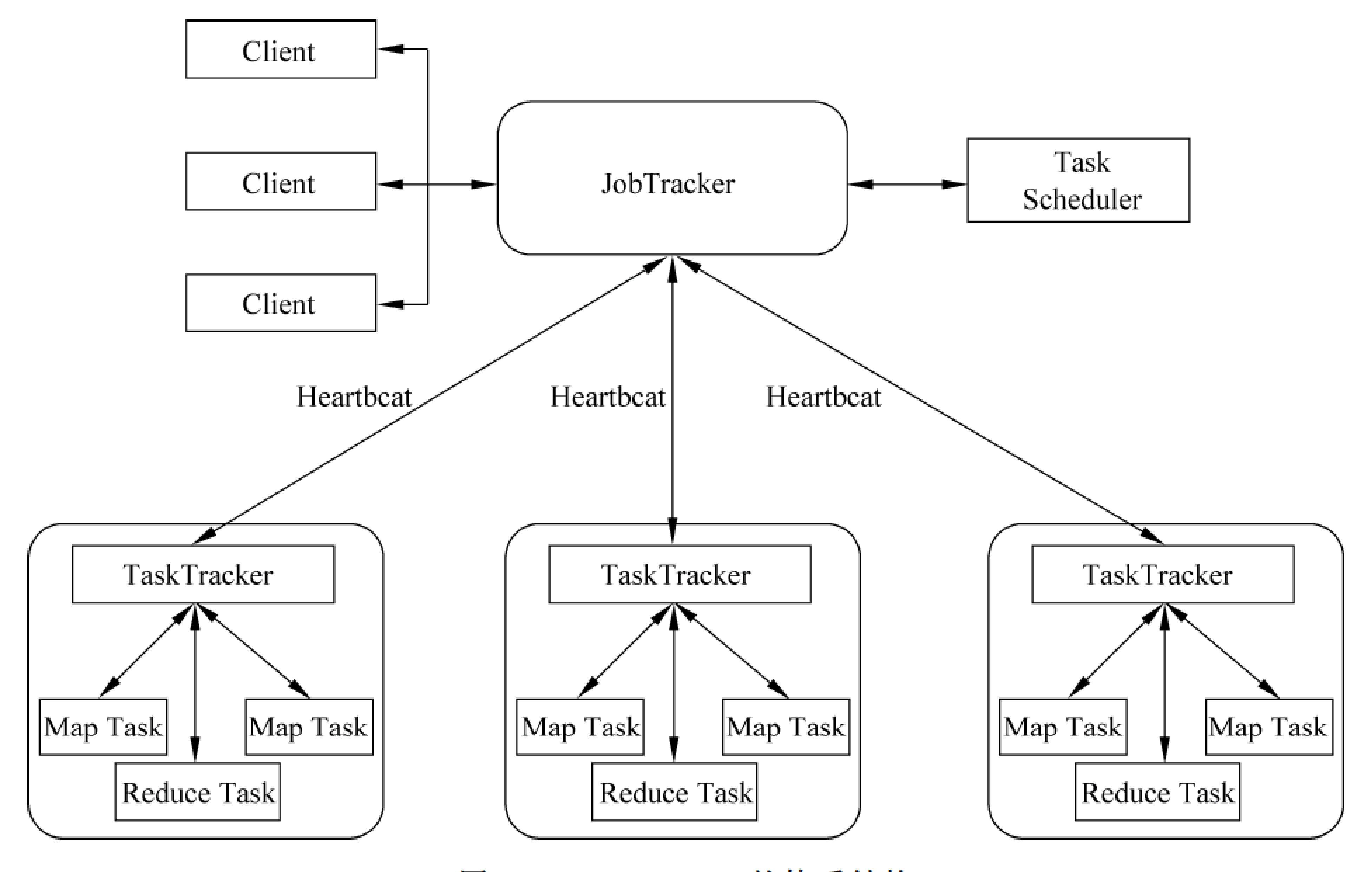


图9-6 MapReduce的体系结构

MapReduce相对于传统的海量数据处理技术而言有着巨大的优势：高效、廉价、弹性、灵活，并且易用。

目前,众多厂商已经把 MapReduce 框架集成到自己的产品或解决方案中。微软的 Big Data Solution、甲骨文的 Oracle Big Data Appliance 等都已经包含或集成了 Hadoop MapReduce 框架。

2. Spark 分布式内存计算处理技术

Spark 是一个分布式的内存计算框架,由加州伯克利大学 AMP 实验室的 Matei 为主的小团队所开发,其特点是能处理大规模数据,计算速度快。Spark 延续了 Hadoop 的 MapReduce 计算模型,但不同于 MapReduce 的是 Job 中间输出和结果可以保存在内存中,从而不再需要读/写 HDFS,因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 Map Reduce 的算法。

在 Spark 中每个作业 Job 被分解成一系列任务 Task,发送到若干个服务器组成的集群上完成。Spark 有分配任务的主结点 Driver 和执行计算的工作结点 Worker。Driver 负责任务分配、资源安排、结果汇总和容错等处理,Worker 负责存放数据和进行计算,如图 9-7 所示。

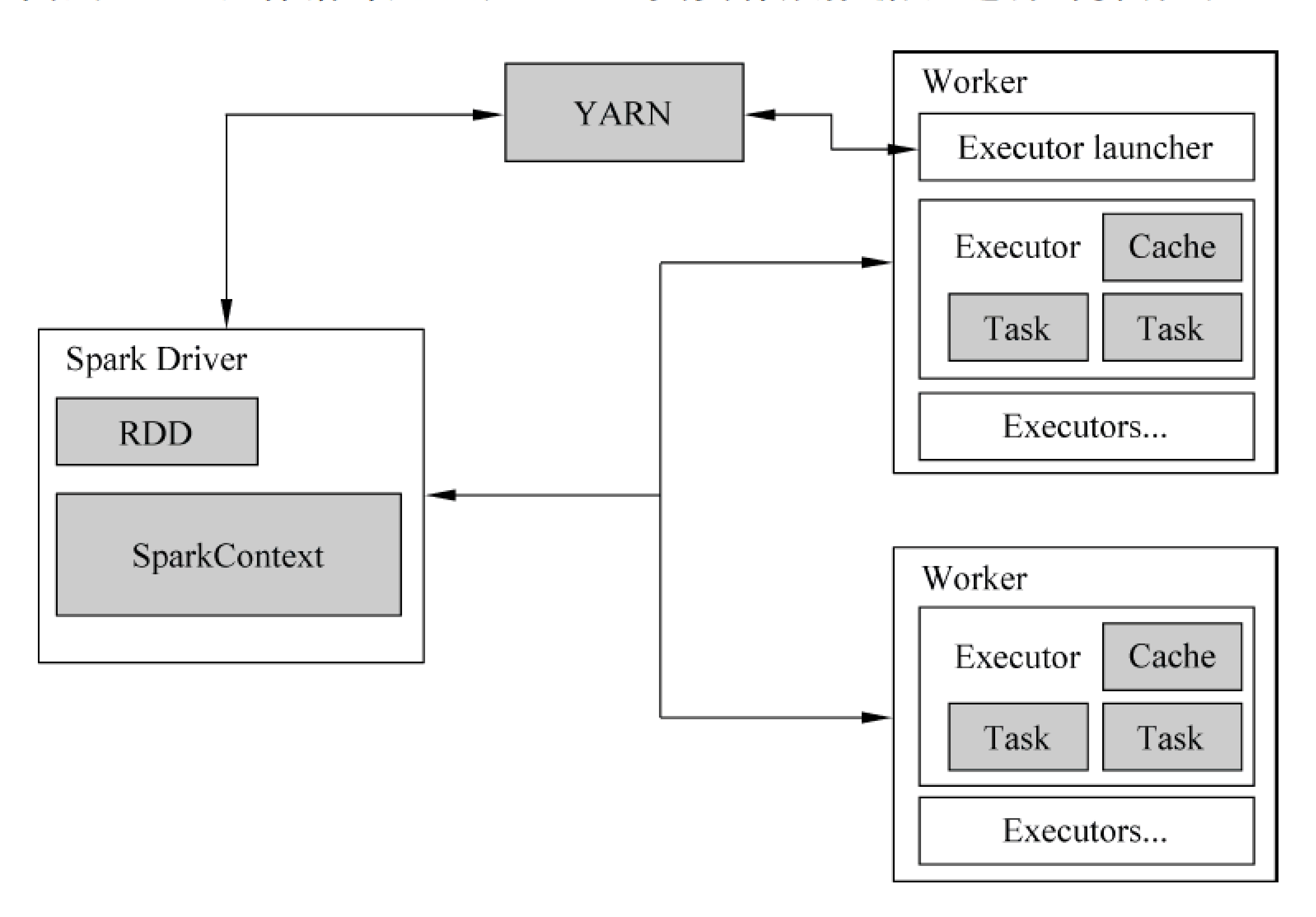


图 9-7　Spark 系统架构图

首先由 Driver 启动若干 Worker,其次 Worker 在分布式的文件系统中读取数据后转换为 RDD(Resilient Distributed Datasets),最后对 RDD 在内存中进行缓存和计算。Spark 将海量数据抽象为 RDD 数据结构,构建起整个 Spark 生态系统,提高数据处理效率。

RDD 是一个容错的、并行的数据结构,可以让用户显式地将数据存储到磁盘和内存中,并能控制数据的分区。同时,RDD 还提供了一组丰富的操作来操作这些数据。在这些操作中,诸如 Map、FlatMap、Filter 等转换操作实现了 Monad 模式,很好地契合了 Scala 的集合操作。除此之外,RDD 还提供了诸如 join、groupBy、reduceByKey 等更为方便的操作(注意,reduceByKey 是 action,而非 transformation),以支持常见的数据运算。

3. Storm 分布式流处理技术

流计算(stream computing)主要针对大量存在的实时数据(流数据),作为一种高实时性的计算模式,流计算需要对一定时间窗口内应用系统产生新数据完成实时的计算处理。典型的应用场景包括证券数据分析、网站广告的上下文分析、社交网络的用户行为分析等。

Storm 实现了一个数据流(data flow)模型,在这个模型中数据持续不断地流经一个由很多转换实体构成的网络。一个数据流的抽象称为流(stream),流是无限的元组(tuple)序列。元组就像一个可以表示标准数据类型(如 int、float 和 byte 数组)和用户自定义类型(需要额外序列化代码)的数据结构。每个流由一个唯一的 ID 来标识,这个 ID 可以用来构建拓扑中各个组件的数据源。

作为管理队列及工作者集群的一种方式,Storm 为分布式实时计算提供了一组通用原语,可被用于"流处理"之中,实时处理消息并更新数据库。Storm 也可被用于"连续计算"(continuous computation),对数据流做连续查询,在计算时就将结果以流的形式输出给用户。它还可被用于"分布式 RPC",以并行的方式运行昂贵的运算。

Storm 主要架构由一个主结点(master node)和一组工作结点(worker nodes)组成,通过 Zookeeper 集群进行协调。主结点通常运行一个后台程序 Nimbus,接收用户提交的任务,并将任务分配到工作结点,同时进行故障监测。工作结点同样会运行一个后台程序 Supervisor,用于接收工作指派并基于要求运行工作进程 Worker。Nimbus 和 Supervisor 之间的协调工作都是通过 Zookeeper 集群来进行的,具体如图 9-8 所示。

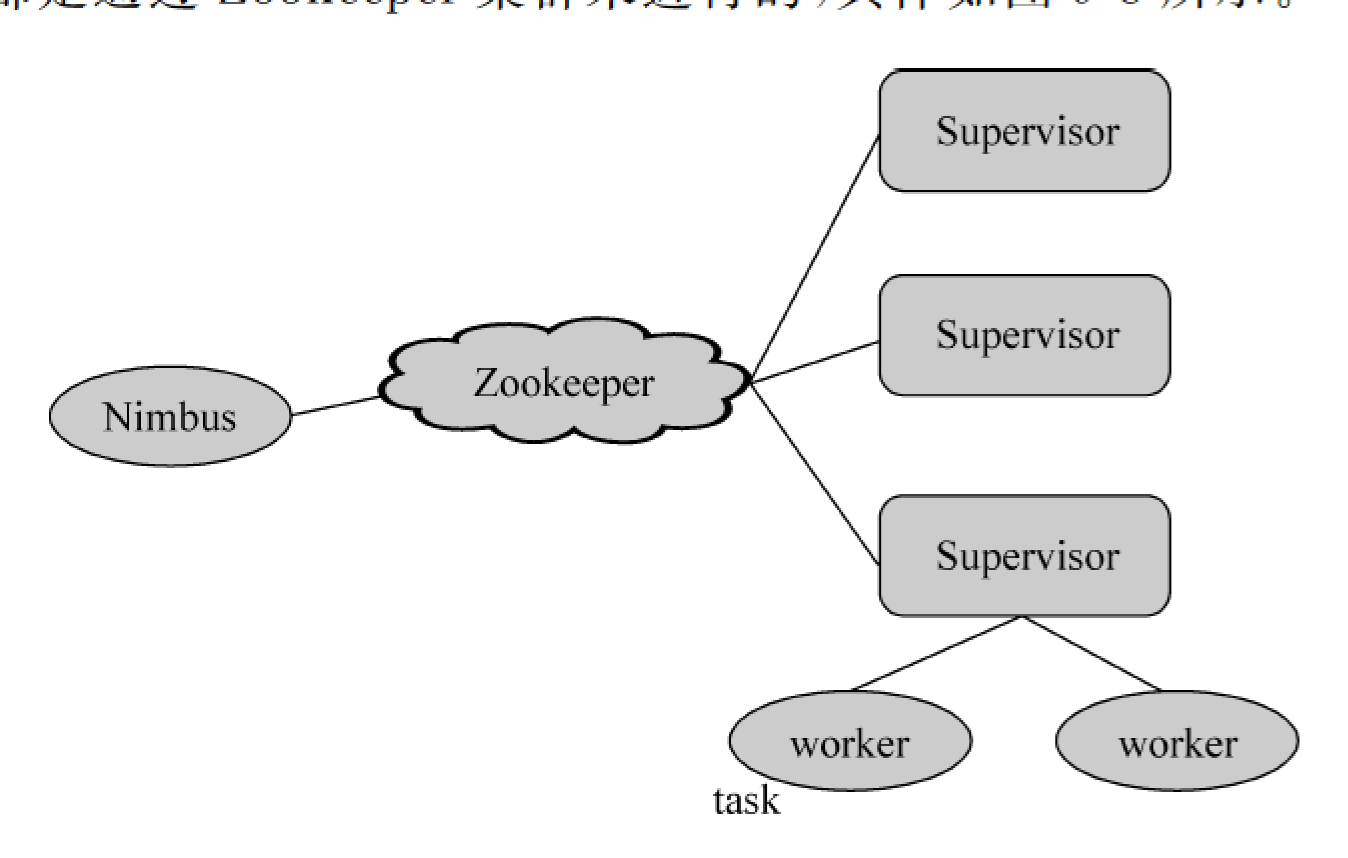


图 9-8　Storm 架构

Storm 比较擅长处理实时的新数据和更新数据库,并兼具容错性和可扩展性;能够进行连续的查询、计算并把结果即时反馈给客户端,并且能够进行分布式远程程序调用,可以用来并行搜索或处理大集合的数据。

9.2.5　大数据分析

物联网、移动互联网再加上常规互联网,每天都在产生体量巨大的各类数据;大数据通过云计算方式将这些数据筛选处理分析,提取出有用的信息,这就是大数据分析。越来越多的应用涉及大数据,而大数据的属性包括数量、速度和多样性等都呈现出大数据不断增长的复杂性,所以大数据分析可看作是决定最终数据是否有价值的关键性环节。

常规数据分析多依靠建立模型的方法,而大数据分析主要是基于规模庞大的数据而使用统计学的方法,其中,语音识别和机器翻译等技术有望在大数据时代取得新的进展。

1. 大数据分析概述

大数据分析包括下述几个基本点。

(1) 可视化分析(analytic visualizations)。不管是对数据分析专家还是普通用户,大数

据分析的使用者最基本的要求都是数据可视化。可视化能够直观呈现大数据特点,容易被理解、接受和使用。

(2) 数据挖掘算法(data mining algorithms)。大数据分析的核心是数据挖掘算法,一方面基于不同的数据类型和格式,采用集群、分割和孤立点分析等算法深入数据内部,更加科学地挖掘出数据本身的价值;另一方面,需要更加快速地处理数据,这也是体现大数据价值的关键所在。

(3) 预测性分析能力(predictive analytic capabilities)。数据分析让人们更好地理解数据,从而能够根据可视化分析和数据挖掘结果做出一些预测性的判断。

(4) 语义引擎(semantic engines)。非结构化数据的多样性带来数据分析的新挑战,人们需要一系列的工具去解析、提取和分析数据语义。语义引擎需要被设计成能够从"文档"中智能提取信息。

(5) 数据质量和数据管理(data quality and master data management)。数据质量和数据管理是大数据分析的基本要求,如此才能保证通过标准化的流程和工具对数据进行处理以得到一个真实的和高质量的分析结果。

大数据分析平台提供大数据云存储与管理、分布式并行数据挖掘任务执行和可视化数据分析等功能,主要由下述模块构成。

① 云计算引擎:支撑海量数据处理、挖掘与分析计算。

② 工作流引擎:包括数据处理分析流程图形化,设计数据处理分析流程,自动执行资源调度和优化。

③ 高性能海量数据挖掘算法库:提供海量复杂数据处理、分析与挖掘的高可扩展算法。

④ 云存储:支撑海量数据存储与管理。

⑤ 开放平台(open API):即开放的应用编程接口,提供数据挖掘平台与第三方应用系统的扩展接口。

大数据分析关键技术源于统计学和计算机科学等多个学科,其核心是能够在不同的数据类型中进行交叉分析。

大数据分析和挖掘的核心部件平台重要组成部分就是高性能海量数据挖掘算法库,此时可采用 MapReduce 等并行处理方式,将巨量数据分解并分布存储,由数据挖掘系统并行处理,并将多个局部处理结果合成最终的输出模式。

大数据挖掘技术的代表有 Hive 和 Mahout。

① Hive:基于 Hadoop 的 PB 级数据仓库平台,在 Hadoop 上管理和查询数据并完成海量数据挖掘。Hive 定义了一个类似于 SQL 的查询语言 HQL,能够将用户编写的 SQL 转换为相应的 MapReduce 任务并运行,习惯于使用 SQL 的用户也能够方便地完成并行计算。

② Mahout:机器学习与数据挖掘算法库,提供一些可扩展的机器学习领域经典算法的实现,如集群、分类和推荐过滤等,与 Hadoop 结合后可以提供分布式数据分析挖掘功能。

常规数据挖掘和大数据时代数据挖掘在分析能力上的差异如下。

首先,常规数据挖掘所处理的数据大多都是关系型数据库中标准化和结构化数据,而对于大数据而言,更多需要处理的是半结构化和非结构化的数据。

其次,大数据处理的数据量与常规处理的数据量完全不在同一个量级之上。常规数据挖掘采用抽样形式对数据进行样本分析,大数据分析则是利用云计算超强的计算处理能力,

对庞大的数据量进行全样本分析，原有的很多数据处理方法已经不能满足需求。

最后，如前所述，应用于决策问题的大数据分析对于数据处理的时效性有着很高的要求。

这些本质上的区别不仅要求大数据使用专门的数据储存技术和设备，还要采用专门的数据分析方法和使用体系。

2. 数据分析和挖掘技术

数据分析与挖掘中的常用技术有关联分析、聚类分析、分类与回归和文本挖掘算法等。

1）关联分析

关联分析是从大量数据中发现项集之间有趣和有意义的关联和相关联系，所发生的联系可以用关联规则或频繁项集的形式表示。

进行关联分析之前，先将数据集看作一个事务的集合，每个事务中包含若干条数据项。一个事务表示一个有意义的单元区间，在这个有意义的单元区间中，若干条数据共同出现即为一个事务。关联分析的主要任务是从事务集中找出数据之间的强关联关系，这些关系的表现形式有两种：频繁项集和关联规则，其中关联规则是最终需要的输出。由于关联规则难以直接从数据集中得到，通常是先从数据中找出关联的另外一种表现形式——频繁项集，然后再将频繁项集转换为关联规则，这是因为通常一起出现的事务之间往往存在一定的关系，寻找关联规则问题就可以转换为寻找经常在一起出现的数据子集。

Apriori 算法和 FP-growth 算法是两种常用的频繁项集发现技术。

(1) Apriori 算法：基于挖掘频繁项集发现布尔关联规则的方法。通过使用频繁项集性质的先验知识，Apriori 算法采取了一种逐层搜索的迭代方法，其中 k 项集用于探索($k+1$)项集。首先，通过扫描数据库，累计每个项的技术，并收集满足最小支持度的项，找出频繁 1 项集的集合，该集合记为 L_1，其次使用 L_1 找出频繁 2 项集的集合 L_2，使用 L_2 找出 L_3，以此类推，直到不能再找出频繁 k 项集，找出每个 L_k 需要一次完整的数据库扫描。最后在所有的频繁项集中提取强规则，即产生用户所感兴趣的关联规则。

(2) FP-growth 算法：主要用于解决发现频繁项集。发现频繁项集可使用 Apriori 算法解决，但此时虽然利用 Apriori 原理加快了发现速度，但其发现效率仍难以满足实际需求。FP-growth 算法使用一种特殊的前缀树——频繁模式树(Frequent Pattern Tree，FP-Tree)。前缀树是一种存储候选项集的数据结构，树的分支用项名标识，树中结点存储后缀项，路径表示项集。FP-Tree 由频繁项头表和项前缀树构成。相对于 Apriori 生成频繁项集，FP-growth 算法生成频繁项集的速度可以快到几个数量级以上。关联挖掘的数据集往往具有很高的重复性，使用 FP-Tree 存储数据还可以减少存储空间，带来比较理想的数据压缩比。

在关联分析技术中，还会针对不同的应用，使用多层和多维关联规则。在许多实际应用中，由于多维数据空间数据的稀疏性，在底层或原始层的数据项之间很难找出强关联规则。在较高层次上挖掘得到的规则可能是更为普通的信息，但对于一些用户也许是非常有价值的信息，这是因为对于不同的用户来说，信息的价值也会不同。由此看来，数据挖掘应该具备在多个层次上进行挖掘的能力，多维关联规则就涉及两个或两个以上层次的关联规则。

2）聚类分析

聚类分析根据“物以类聚”的道理，对样本或指标进行分类的一种多元统计分析方法，它也是数据挖掘和模式识别等研究中的基本内容之一，在识别数据的内在结构方面具有重要

作用。聚类分析旨在发现紧密相关的观测值群组,使得与属于不同簇的观测值相比,属于同一簇的观测值相互之间尽可能相似。

聚类分析算法可分为划分方法(partitioning methods)、层次方法(hierarchical methods)、基于密度的方法(density-based methods)、基于网格的方法(grid-based methods)和基于模型的方法(model-based methods)。以下简单介绍划分方法和层次方法。

(1) 划分聚类:对于一个给定的包含 n 个数据对象的数据库,要把其中的对象分为 k 个聚类,划分方法就是运用一些相关的算法将对象集合划分成 k 份,其中每个划分表示一个聚类。其中要求属于一个聚类中的对象是相似的,属于不同聚类的对象是不相似的。常用划分聚类包括 K-means、K-Medoids、EM 算法等。

(2) 层次聚类:将数据分为若干组并形成一个组的树从而进行聚类,一般分为自下而上聚合层次聚类方法 AGNES(AGglomerative NESting)和自顶而下分解层次聚类方法 DIANA(DIvsia ANAlysia)。AGNES 通常将每个对象构成一个单独聚类,然后根据一定标准不断进行聚合。例如,对于聚类 C1 和 C2 来讲,若 C1 中对象与 C2 中对象之间的欧式距离为不同聚类中任两个对象之间的最小距离,则聚类 C1 和 C2 就可以进行聚合。两个聚类之间相似程度是利用相应两个聚类中每个对象之间的最小距离来加以描述的。AGNES 方法不断进行聚合操作,直到所有聚类最终聚合为一个聚类为止。DIANA 方法首先是将所有的对象集中在一起构成了一个聚类。其次根据一定原则,如聚类中最近对象之间的最大欧式距离等对其进行不断分解,直到每个聚类均只包含一个对象为止。层次聚类算法能够产生高质量的聚类,但也存在计算和存储需求较大、缺乏全局的目标函数和合并决策不能撤销等问题。

聚类算法的选择取决于数据类型、聚类目的和实际应用。当将聚类分析用作描述或探查的工具时,可以对同样的数据尝试使用多种算法,以发现数据可能揭示的结果。

3) 分类与回归

分类与回归本质上是两种不同的预测方法。

(1) 分类:主要是预测离散的目标变量,输出的是离散值。

(2) 回归:主要用于预测连续的目标变量,输出的是有序值或连续值。

分类问题实际上是要建立一个从输入数据到分类标签的映射。通过机器学习技术可以建立起相应的映射模型,即使用某种学习算法,按照一定策略对输入数据进行分析,找出一个能够很好地拟合输入数据和数据类标号的映射,而该映射还能够正确地预测未知数据和它的类标号。

回归分析实际上是要确定两种或两种以上变量之间相互依赖的定量关系的统计分析方法。回归分析应用十分广泛,按照涉及自变量个数多少可分为一元回归分析和多元回归分析;按照自变量和因变量之间关系类型可分为线性回归分析和非线性回归分析。回归的主要目的是预测数值型数据,最直接的方法是根据驶入数值做出一个计算目标值的公式也就是回归方程。回归方法基本点是优化回归方程中的参数,从而减少回归预测误差。

4) 文本挖掘算法

文本挖掘(text mining)由非结构化文本信息中获取用户感兴趣或有用模式的过程,其本质是从大量文本数据中抽取事先未知的、可理解的和最终可用的知识,同时使用得到的相应知识更好地组织信息以便将来参考使用。

文本挖掘主要用到的工具包括文本分词、文本挖掘和语义分析等。

5）数据可视化技术

大数据可视分析以“人”作为分析主体和需求主体角度出发，旨在利用计算机自动化分析能力的同时，充分挖掘“人”对于可视化信息的认知能力优势，将人、机的各自强项进行有机融合。借助人机交互式分析方法和交互技术，根据不同用户的需求对挖掘结果进行处理并以图形化手段等直观的方式进行展现，从而通过数据可视化手段辅助人们更为直观和高效地洞悉大数据背后的信息、知识与智慧。

据统计，人类从外界获得的信息约有 80%以上来自于视觉系统，当大数据以直观的可视化的形式展示在分析者面前时，分析者往往能够“一眼洞悉”数据背后隐藏的信息。例如，互联网星际图将 196 个国家的 35 万个网站数据整合起来，并根据 200 多万个网站链接将这些互联网“星球”通过关系链联系起来，每一个“星球”的大小根据其网站流量来决定，而“星球”之间的距离远近则根据链接出现的频率、强度和用户跳转时创建的链接。人们往往立即就可看出 Facebook 及 Google 是流量最大的网站。这些“一眼”识别出的图形特征（如异常点和相似的图形标记等）在通过视觉就能够相当容易地被察觉和接受，而通过机器计算却很难理解其含义。由此可知，大数据可视分析是大数据分析不可或缺的重要手段和工具。

数据可视化（data visualization）是指应用计算机图形学和图像处理技术，将数据转换为图形或图像在屏幕上显示出来，并进行交互处理的理论、方法和技术。作为可视化技术在非空间数据领域的应用，它使得人们不再局限于通过关系数据表来观察和分析数据信息，而是以更为直观的方式观察各类数据和相互之间的结构关系。

数据可视化技术基本思想：将数据库中每个数据项表示为单个图元元素，大量的数据集最终构成整体的数据图像，同时将数据的各个属性值以多维数据的形式表示，便于人们从不同的维度观察数据，从而对数据进行更深入的观察和分析。

数据可视化技术涉及计算机图形学、图像处理、计算机辅助设计、计算机视觉及人机交互技术等多个领域。传统的可视化交互通常基于电子表格做出的数字列表，或者以柱状图、饼状图等简单的图形化展示数据形态。随着大数据时代的来临，数据之间蕴含着更深层次的关联和某些隐藏的规律，当需要深入洞察数据，则需要更先进的、更多维度的可视化和交互技术，其中包括 2D 展示技术、3D 渲染技术、体感互动技术、虚拟现实技术、增强现实技术、可穿戴技术和可植入设备等。

9.3 MongoDB 概述

MongoDB 是由 10gen 公司开发的一款由 C++语言编写的基于分布式文件存储的数据库。2012 年 5 月发布 MongoDB2.1，2015 年 6 月发布 MongoDB 3.0.4。

作为一种介于 RDB 和非 RDB 之间的产品，MongoDB 也许是各类非 RDB 中功能丰富并且更“像” RDB 的开源软件。

（1）MongoDB 优势：体现在提高体量巨大数据的访问效率，为 Web 应用提供可扩展的高性能数据存储解决方案。根据统计，当数据量达到 50GB 以上时，MongoDB 数据库访问速度是 MySQL 的 10 倍以上。MongoDB 自带分布式文件系统 GridFS 用以支持大数据存储。

（2）MongoDB 特征：无表结构，所支持的数据结构相当松散，不需要设置相同的字段，并且相同的字段不需要相同的数据类型，由于不需要定义表结构，减少了添加字段等表结构

变更所需要的开销。

(3) MongoDB 查询语言：查询语言的语法类似于面向对象的数据查询语言，几乎可以实现类似 RDB 单表查询的绝大部分功能，同时支持对数据建立索引。作为文档型数据库，由于数据的处理方式不同，MongoDB 的用语有着自身的特点，如 RDB 中的“关系表”和“元组(记录)”在 MongoDB 中分别称为“集合”和“文档”。

(4) MongoDB 并发读/写效率：这方面还不够特别出色，根据官方提供的性能测试表明，大约每秒可以处理 5000～15 000 次读/写请求。

MongoDB 服务端可运行在 Linux 9、Windows 或 OS 9 平台，支持 32 位和 64 位应用，默认端口为 27017。推荐运行在 64 位平台，因为 MongoDB 在 32 位模式运行时支持的最大文件尺寸为 2GB。

MongoDB 把数据存储在文件中(默认路径为：/data/db)，为提高效率使用内存映射文件进行管理。

以下简要介绍 MongoDB 的安装与使用。

9.3.1 Windows 下安装 MongoDB

第一步：从 MongoDB 官网(https://www.mongodb.com/download-center)上下载最新的 MongoDB 的预编译二进制包，根据不同的计算机环境配置选择相应的安装包，如图 9-9 所示。下载的是 Community Server 3.27 版本。

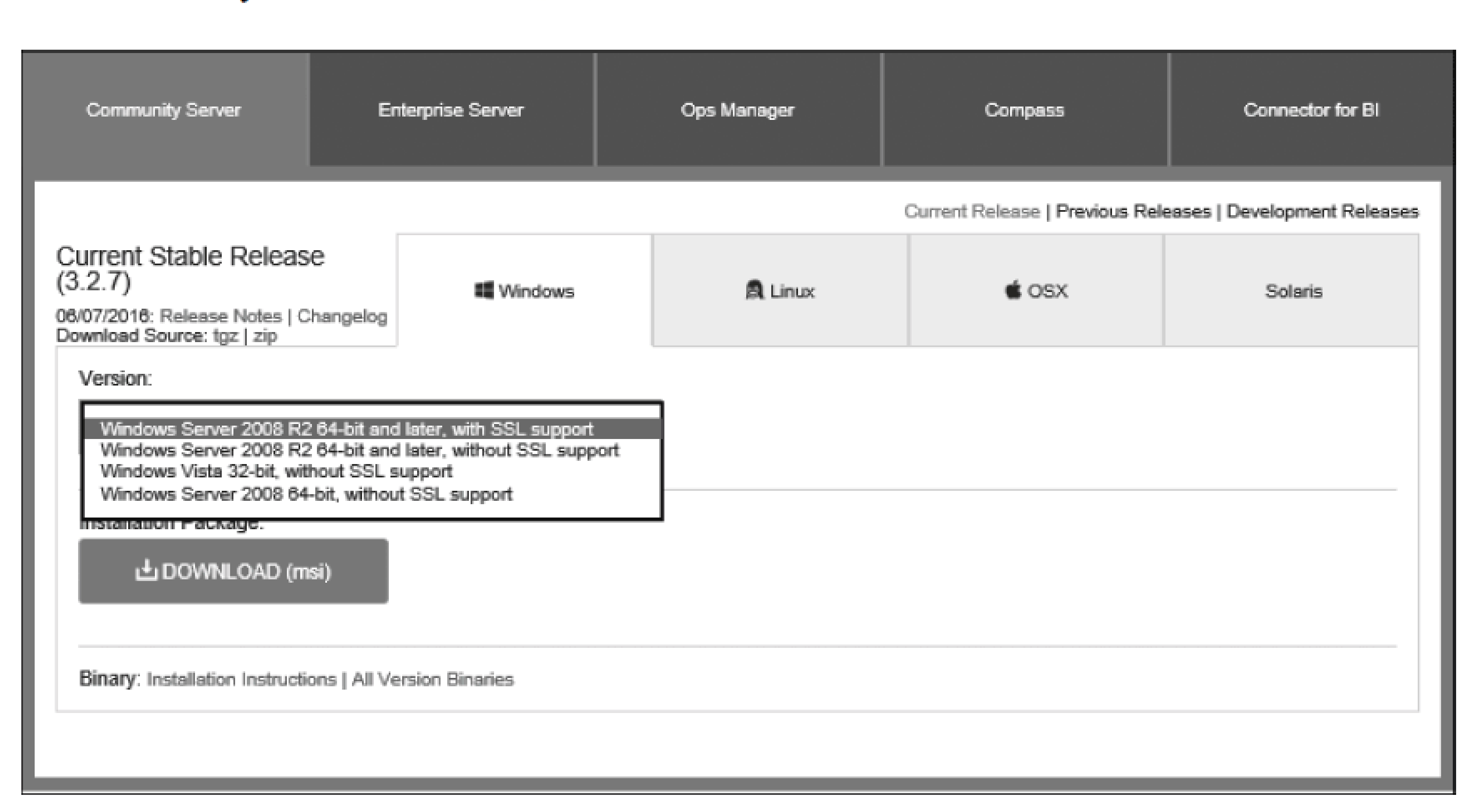


图 9-9　下载 MongoDB

(1) Windows Server 2008 R2 64-bit and later, with SSL support：适用于 64 位并支持 SSL(Secure Socket Layer)的 Windows 系统。

(2) Windows Server 2008 R2 64-bit and later, without SSL support：适用于 64 位且不支持 SSL(Secure Socket Layer)的 Windows 系统。

(3) Windows Vista 32-bit, without SSL support：适用于 32 位且不支持 SSL 的 Windows 系统。

(4) Windows Server 2008 64-bit, without SSL support：适用于 32 位且不支持 SSL 的 Windows 系统。

注意，在 MongoDB2.2 版本后已经不再支持 Windows 9P 系统。

第二步：根据系统下载相应版本的安装包，下载后运行该安装包，如图 9-10 所示。

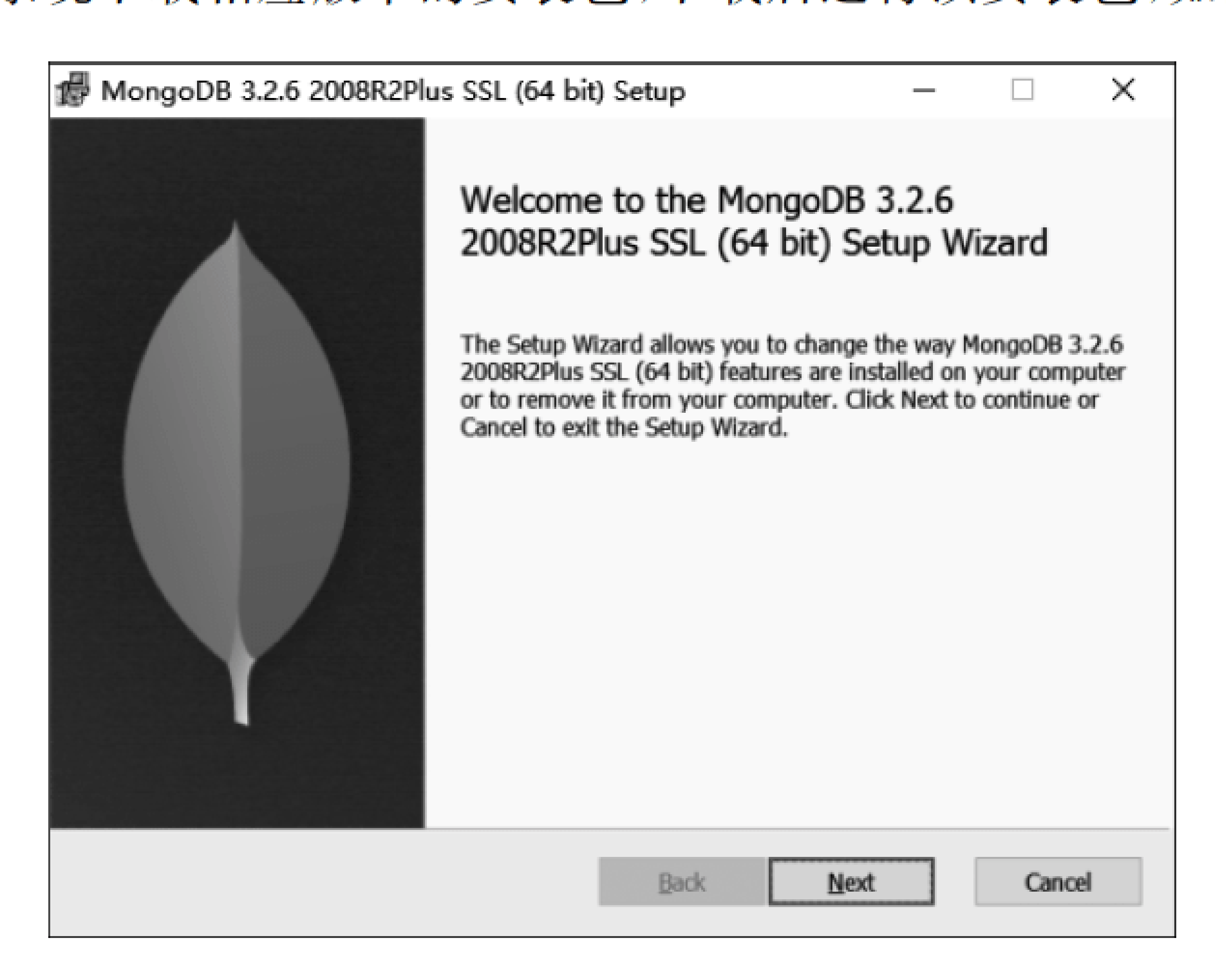


图 9-10　运行 MongoDB 安装包

通过单击 Custom(自定义)按钮来设置安装目录，如图 9-11 所示。

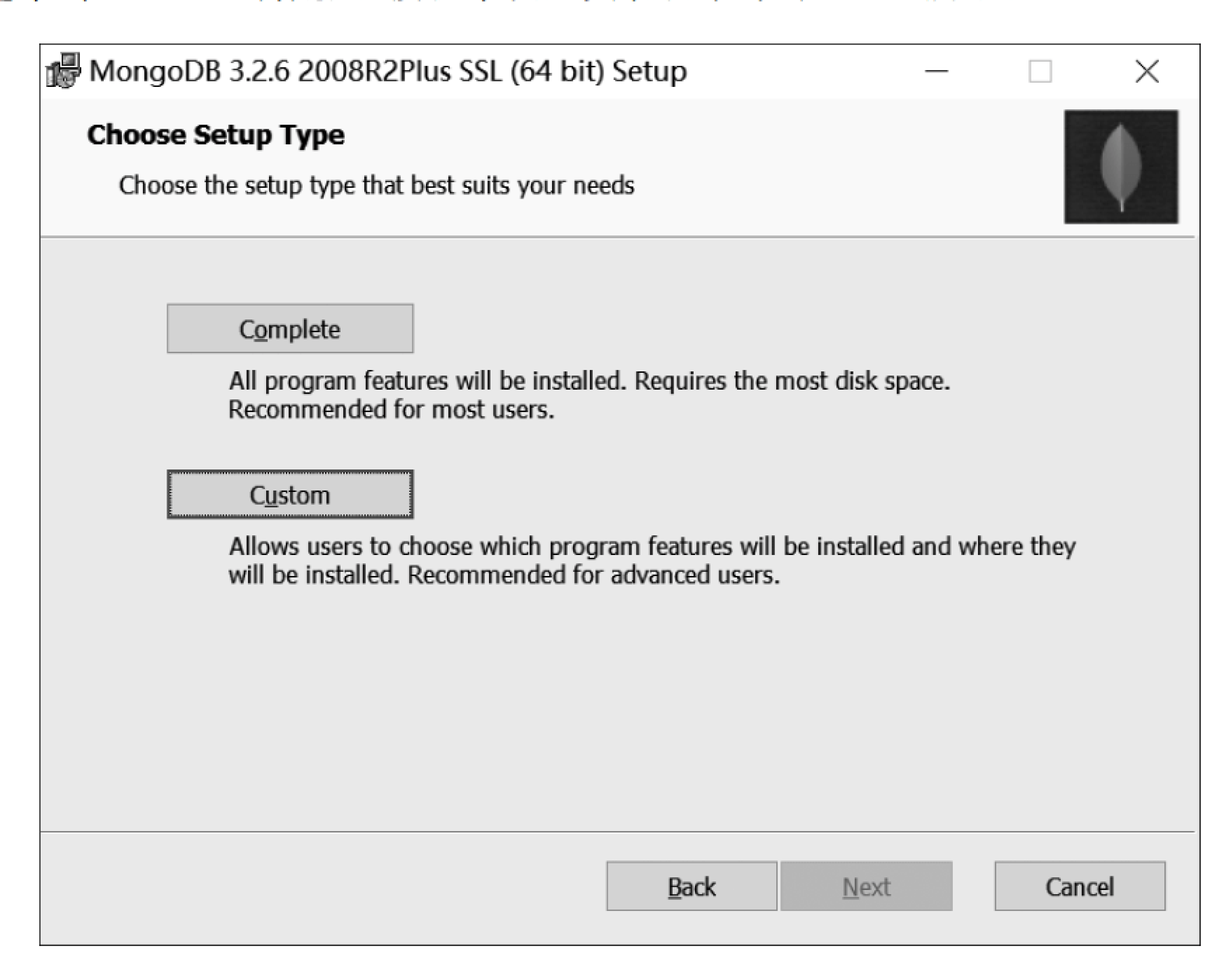


图 9-11　选择 Custom 安装方式

9.3.2　MongoDB 运行环境设置

1. 运行 MongoDB 服务器

方法一：通过命令行的方式运行。

通过从 MongoDB 目录的 bin 目录中执行 mongod.e9e 程序。先进入 bin 文件夹“C:\> cd

c:\ MongoDB \ bin \"，再输入"Mongod. e9e-dbpath c:\ MongoDB \ data \ db"命令运行 MongoDB 服务器。如果执行成功，会输出如图 9-12 所示的信息。

```
d=3308 port=27017 dbpath=c:\MongoDB\data\db 64-bit host=home-school
2016-06-19T00:14:22.752+0800 I CONTROL  [initandlisten] targetMinOS: Windows
7/Windows Server 2008 R2
2016-06-19T00:14:22.753+0800 I CONTROL  [initandlisten] db version v3.2.6
2016-06-19T00:14:22.754+0800 I CONTROL  [initandlisten] git version: 05552b56
2c7a0b3143a729aaa0838e558dc49b25
2016-06-19T00:14:22.755+0800 I CONTROL  [initandlisten] OpenSSL version: Open
SSL 1.0.1p-fips 9 Jul 2015
2016-06-19T00:14:22.756+0800 I CONTROL  [initandlisten] allocator: tcmalloc
2016-06-19T00:14:22.757+0800 I CONTROL  [initandlisten] modules: none
2016-06-19T00:14:22.766+0800 I CONTROL  [initandlisten] build environment:
2016-06-19T00:14:22.767+0800 I CONTROL  [initandlisten]     distmod: 2008plus
-ssl
2016-06-19T00:14:22.768+0800 I CONTROL  [initandlisten]     distarch: x86_64
2016-06-19T00:14:22.769+0800 I CONTROL  [initandlisten]     target_arch: x86_
64
2016-06-19T00:14:22.771+0800 I CONTROL  [initandlisten] options: { storage: {
 dbPath: "c:\MongoDB\data\db" } }
```

图 9-12　运行 MongoDB 服务器

用浏览器访问 http://localhost：27017/会在浏览器界面出现"It looks like you are trying to access MongoDB over HTTP on the native driver port."信息。

方法二：将 MongoDB 作为 Windows 服务运行。

本操作需要有管理员权限才可以执行以下命令。进入 C:\MongoDB\bin\目录下，输入以下命令：

```
mongod.exe -- logpath "C:\MongoDB\data\log\mongodb.log" -- dbpath "C:\MongoDB\data\db" --
serviceName "MongoDB" - install
```

通过执行"运行"→services.msc 命令，可以查看到添加了一个 MongoDB 服务，如图 9-13 所示。

MongoDB	MongoDB Server	正在运行	自动	本地系统

图 9-13　创建 MongoDB 服务

将其设置为手动并启动，可随开机启动，免去了每次通过命令行的方式运行程序。

如表 9-2 所示为 MongoDB 启动的参数说明。

表 9-2　MongoDB 启动的参数说明

参　　数	描　　述
--bind_ip	绑定服务 IP，若绑定 127.0.0.1，则只能本机访问，不指定默认本地所有 IP
--logpath	MongoDB 日志文件路径，注意是指定文件不是目录
--logappend	使用追加的方式写日志
--dbpath	指定数据库路径
--port	指定服务端口号，默认端口为 27017
--serviceName	指定服务名称
--serviceDisplayName	指定服务名称，有多个 MongoDB 服务时执行
--install	指定作为一个 Windows 服务安装

2. 创建数据目录

假设 MongoDB 的安装目录设置为“C:\MongoDB\”,MongoDB 将数据目录存储在 db 文件夹下。例如,在 MongoDB 的安装目录下创建 data 文件夹,并在该文件夹下创建 db 目录。注意,data 文件夹、db 文件夹不会自动创建,需要自行创建文件夹。也可以通过命令行的方式创建 db 目录。

```
c:\> cd c:\MongoDB\
c:\MongoDB > mkdir data
c:\MongoDB > cd data
c:\MongoDB \data > mkdir db
c:\MongoDB \data > cd db
c:\MongoDB\data\db >
```

3. MongoDB 后台管理 Shell

MongoDB Shell 是 MongoDB 自带的交互式 JavaScript shell,用来对 MongoDB 进行操作和管理的交互式环境。

进入 MongoDB 后台后,默认会链接到 test 文档(数据库):

```
> mongo
MongoDB shell version: 3.2.6
connecting to: test
```

9.3.3 可视化管理软件——Robomongo

Robomongo 是一个基于 Shell 的跨平台开源 MongoDB 管理工具。嵌入了 JavaScript 引擎和 MongoDBmogo。只要会使用 Mongo shell,就会使用 Robomongo。提供语法高亮、自动完成、差别视图等可视化模式。如图 9-14 所示为 Robomongo 的界面。

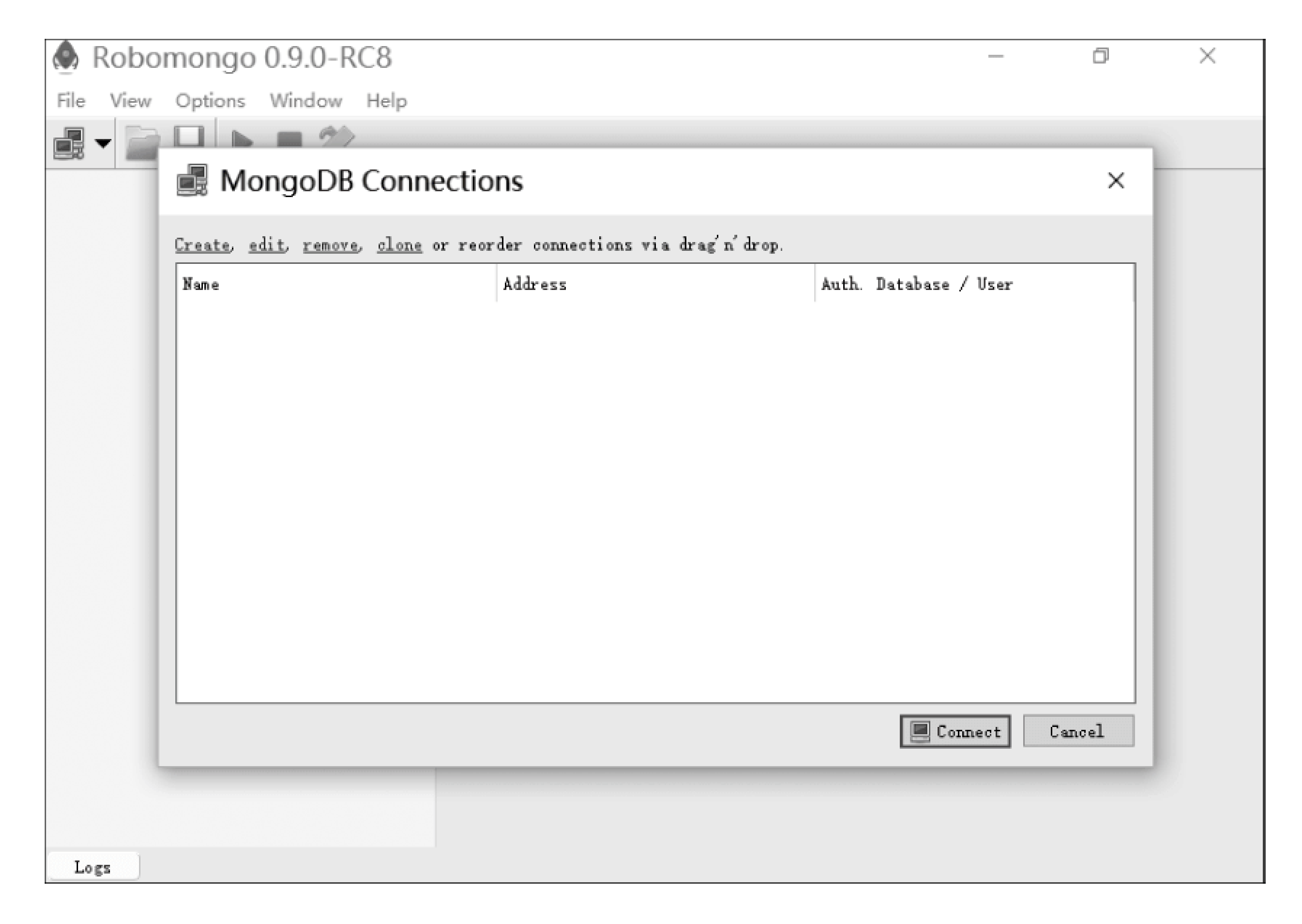


图 9-14 Robomongo 界面

1. 建立 MongoDB 数据库服务器连接

先使用对话框中的 Create 按钮，建立一个 MogoDB 数据库服务器的连接，如图 9-15 所示。

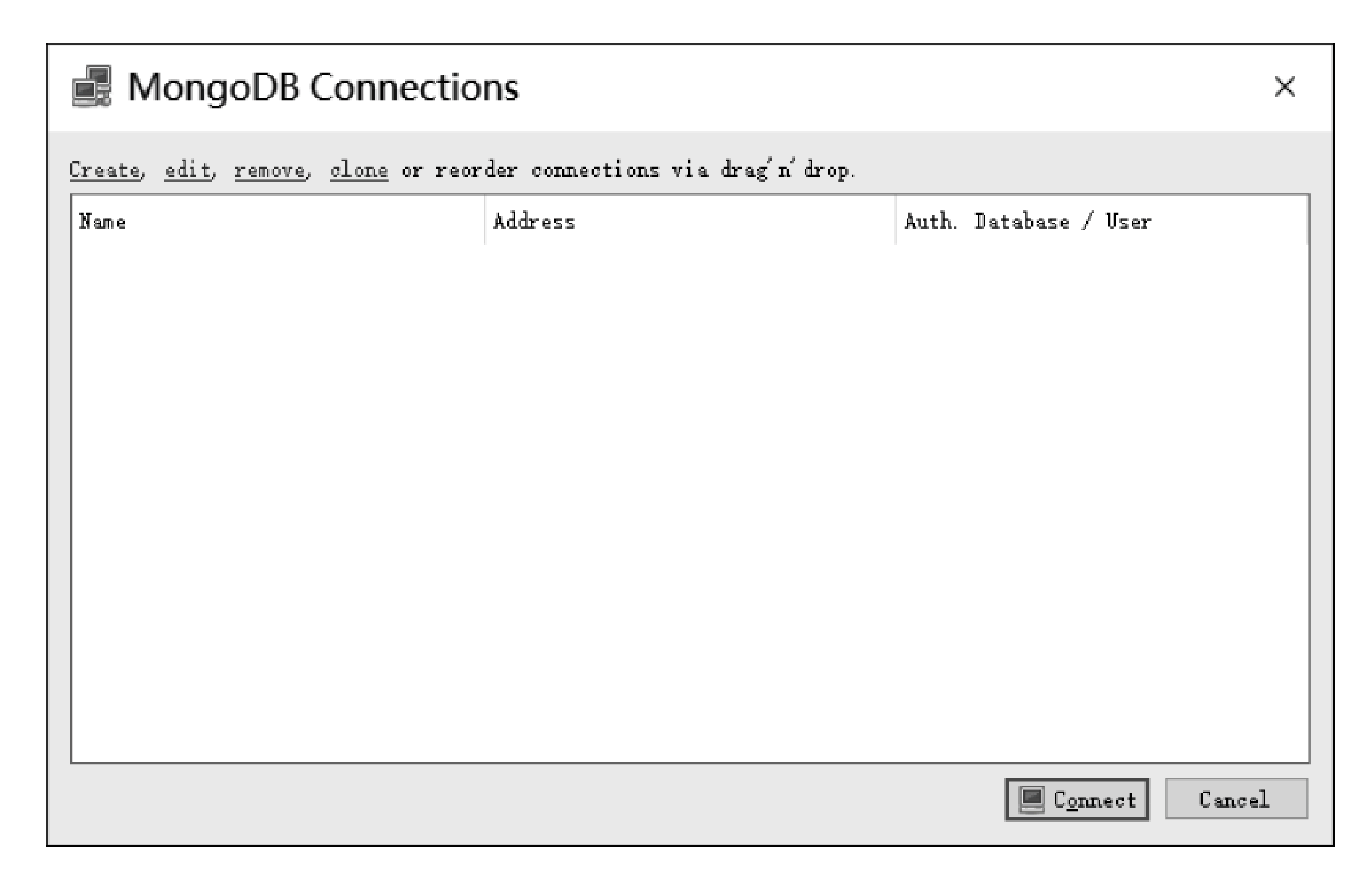


图 9-15　建立 MongoDB 数据库服务器连接

因为本地的 MongoDB 数据库服务器地址默认是 localhost：27017（可以在 MongoDB shell 中更改）执行后的结果如图 9-16 所示。

保存后的界面如图 9-17 所示。

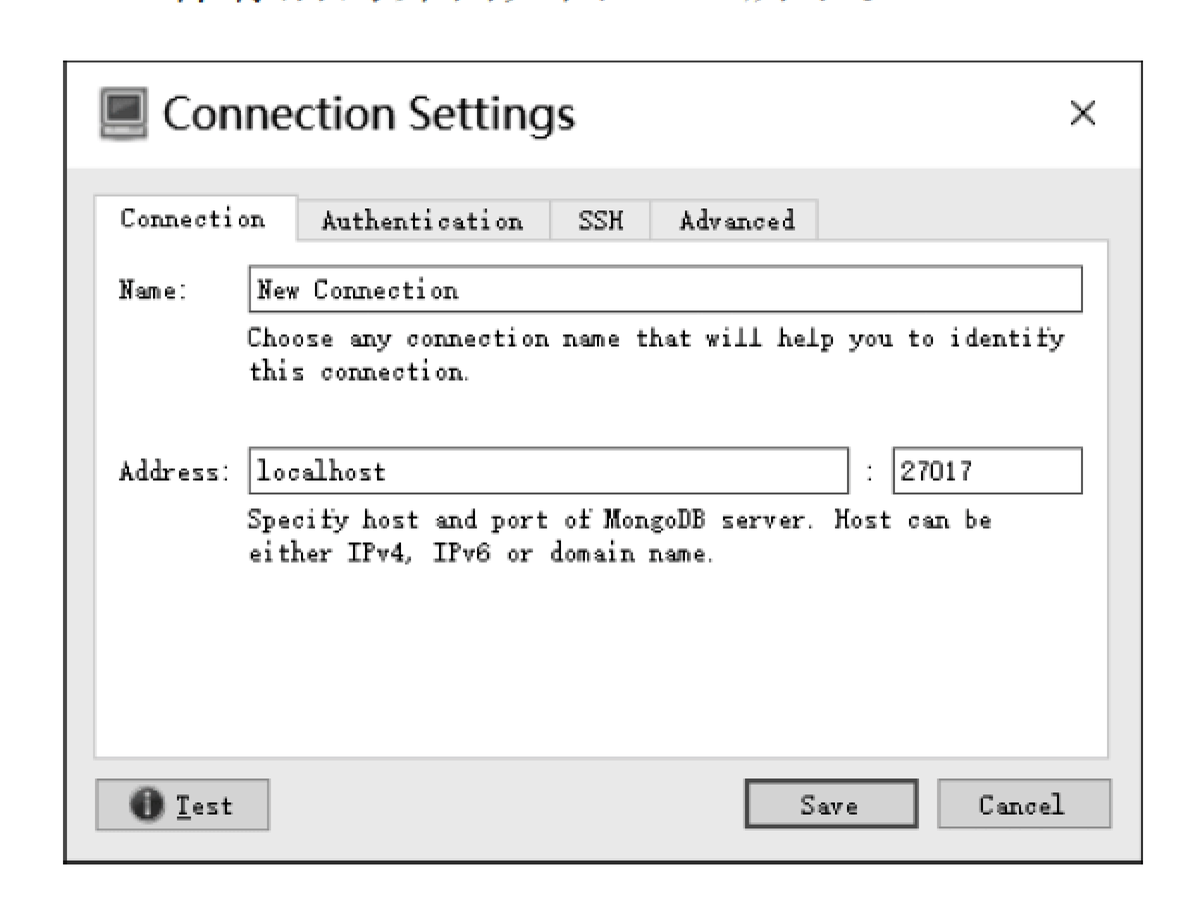


图 9-16　创建连接

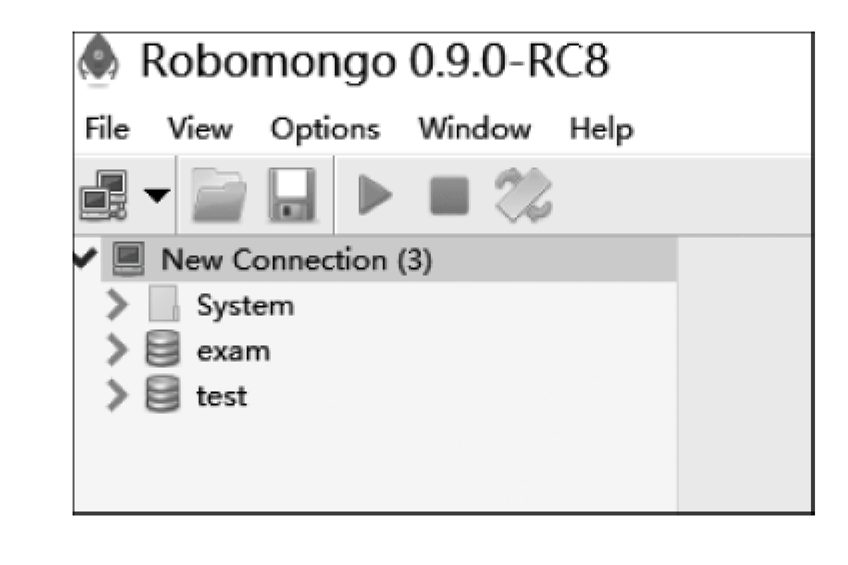
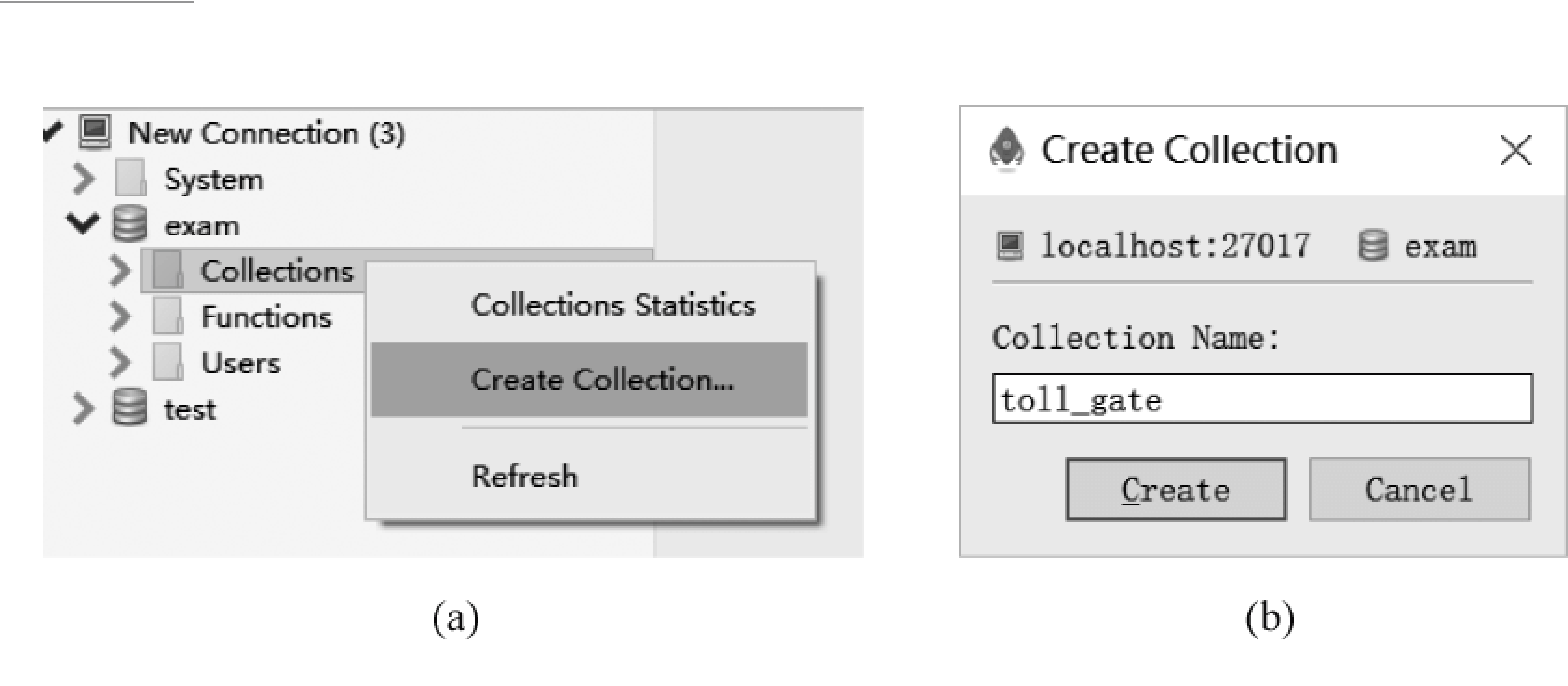


图 9-17　成功连接 MongoDB 数据库

2. 使用 Robomongo 创建新集合

进入数据库 e9am(这个数据库是之前创建的)，然后再创建一个集合 toll_gate，如图 9-18 所示。

创建完之后进入集合 toll_gate，由于集合中没有数据，因此在界面的右方并没有数据的显示，如图 9-19 所示。



(a) (b)

图 9-18 创建集合

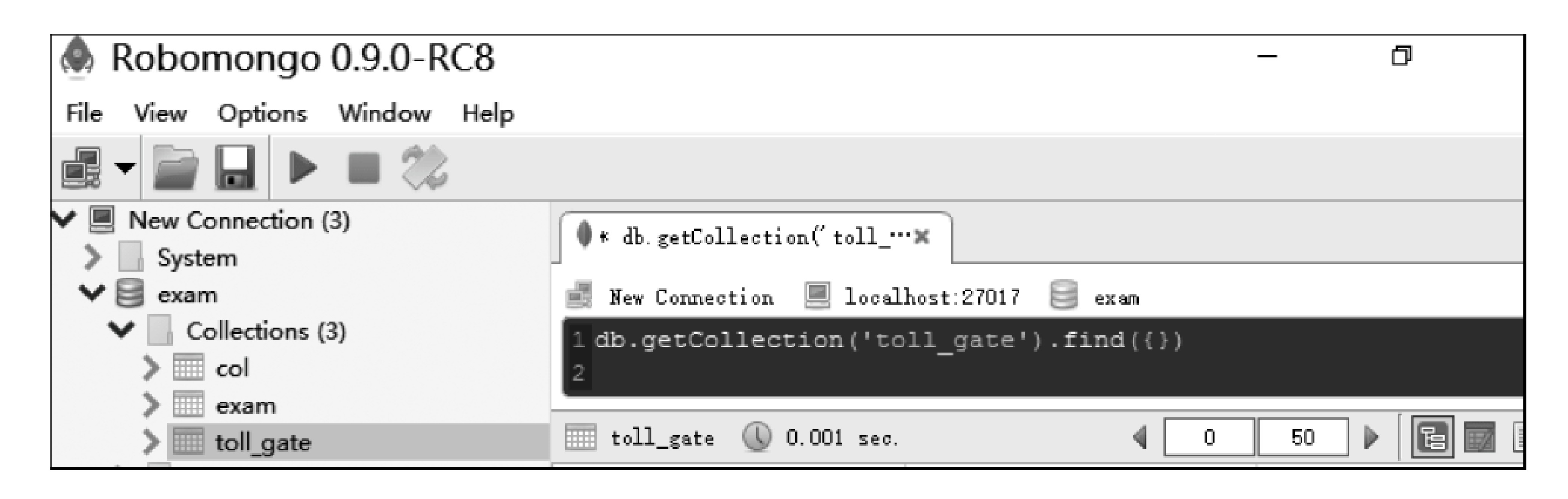


图 9-19 查询集合 toll_gate

3. 使用 Robomongo 插入文档

在此集合中插入一份文档，其具体操作如图 9-20 所示。

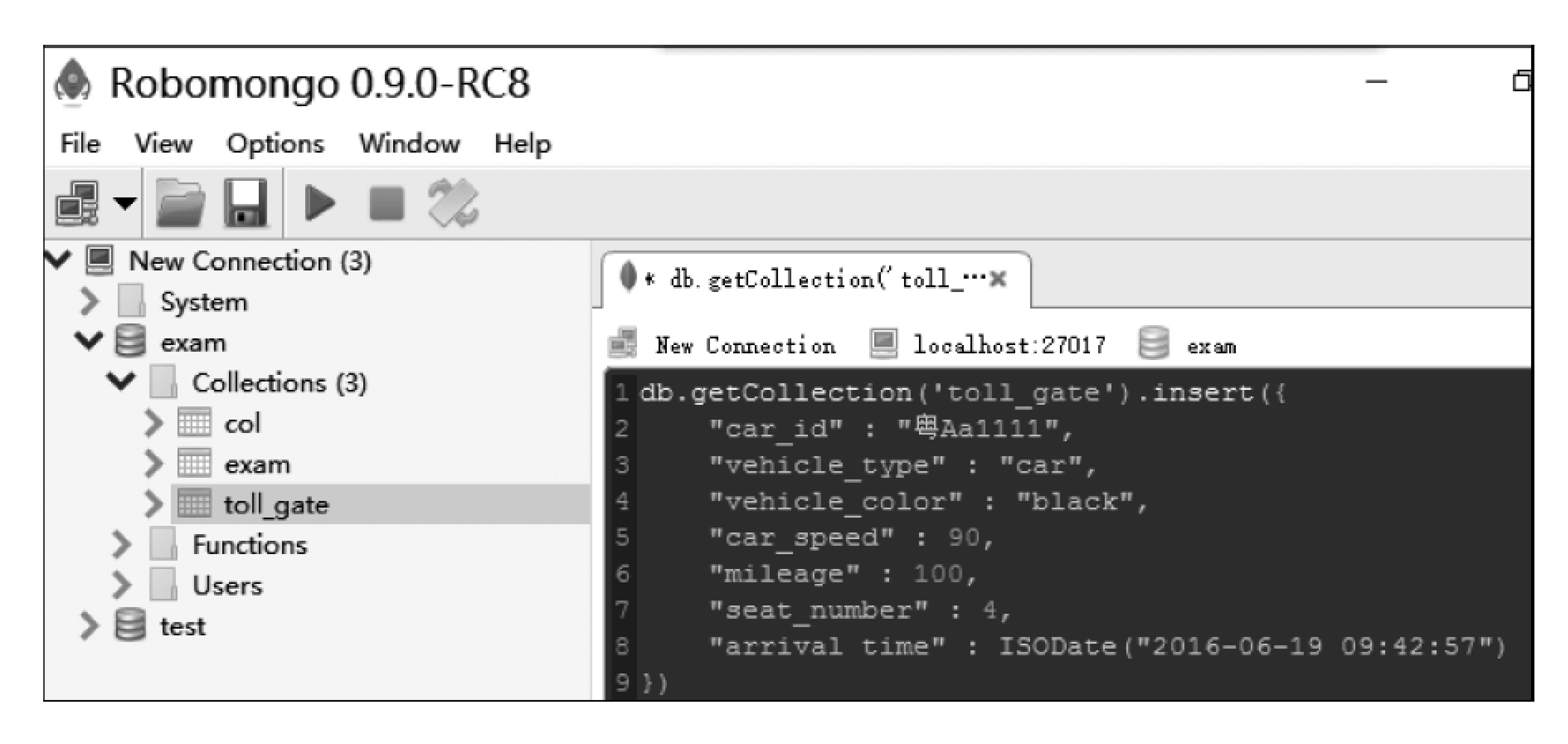


图 9-20 插入文档

检查一下文档是否插入成功，成功的结果如图 9-21 所示。

现在添加 10 个文档如图 9-22 所示。

保存之后显示结果如图 9-23 所示，表示添加文档成功。

4. 使用 Robomongo 查询文档

现在从众多文档中查找 arrival time 的值为 2016-06-12 的文档。如图 9-24 所示，在命令行中输入查询命令，按 Ctrl+Enter 组合键执行。

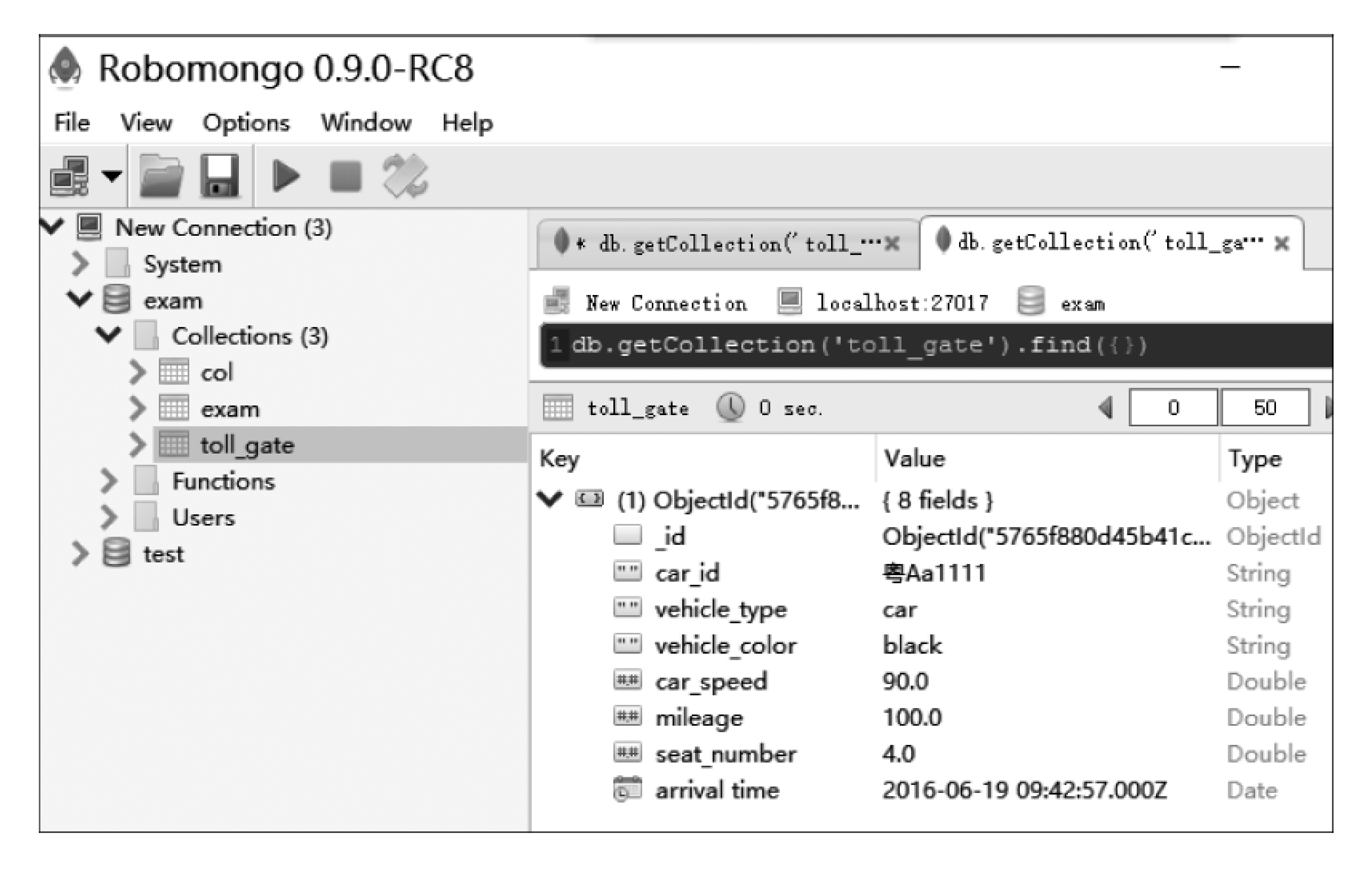


图 9-21　成功插入文档

New Connection　localhost:27017　exam

```
    "car_id" : "粤Aab111",
    "vehicle_type" : "truck",
    "vehicle_color" : "black",
    "car_speed" : "90",
    "mileage" : "100",
    "seat_number" : "4",
    "arrival time" : ISODate("2016-06-09 12:31:40")
})
db.getCollection('toll_gate').insert(
{
    "car_id" : "粤Aag666",
    "vehicle_type" : "car",
    "vehicle_color" : "gray",
    "car_speed" : "121",
    "mileage" : "134",
    "seat_number" : "4",
    "arrival time" : ISODate("2016-06-12 07:20:25")
})
```

图 9-22　添加 10 个文档

Key	Value	Type
(5) ObjectId("5765f...	{ 3 fields }	Object
(6) ObjectId("5765f...	{ 6 fields }	Object
(7) ObjectId("5765f...	{ 8 fields }	Object
(8) ObjectId("5765f...	{ 7 fields }	Object
(9) ObjectId("5765f...	{ 8 fields }	Object
(10) ObjectId("5765...	{ 8 fields }	Object
(11) ObjectId("5765...	{ 8 fields }	Object
_id	ObjectId("5765ff68d45b41...	ObjectId
car_id	粤Aag666	String
vehicle_type	car	String
vehicle_color	gray	String
car_speed	121	String
mileage	134	String
seat_number	4	String
arrival time	2016-06-12 07:20:25.000Z	Date

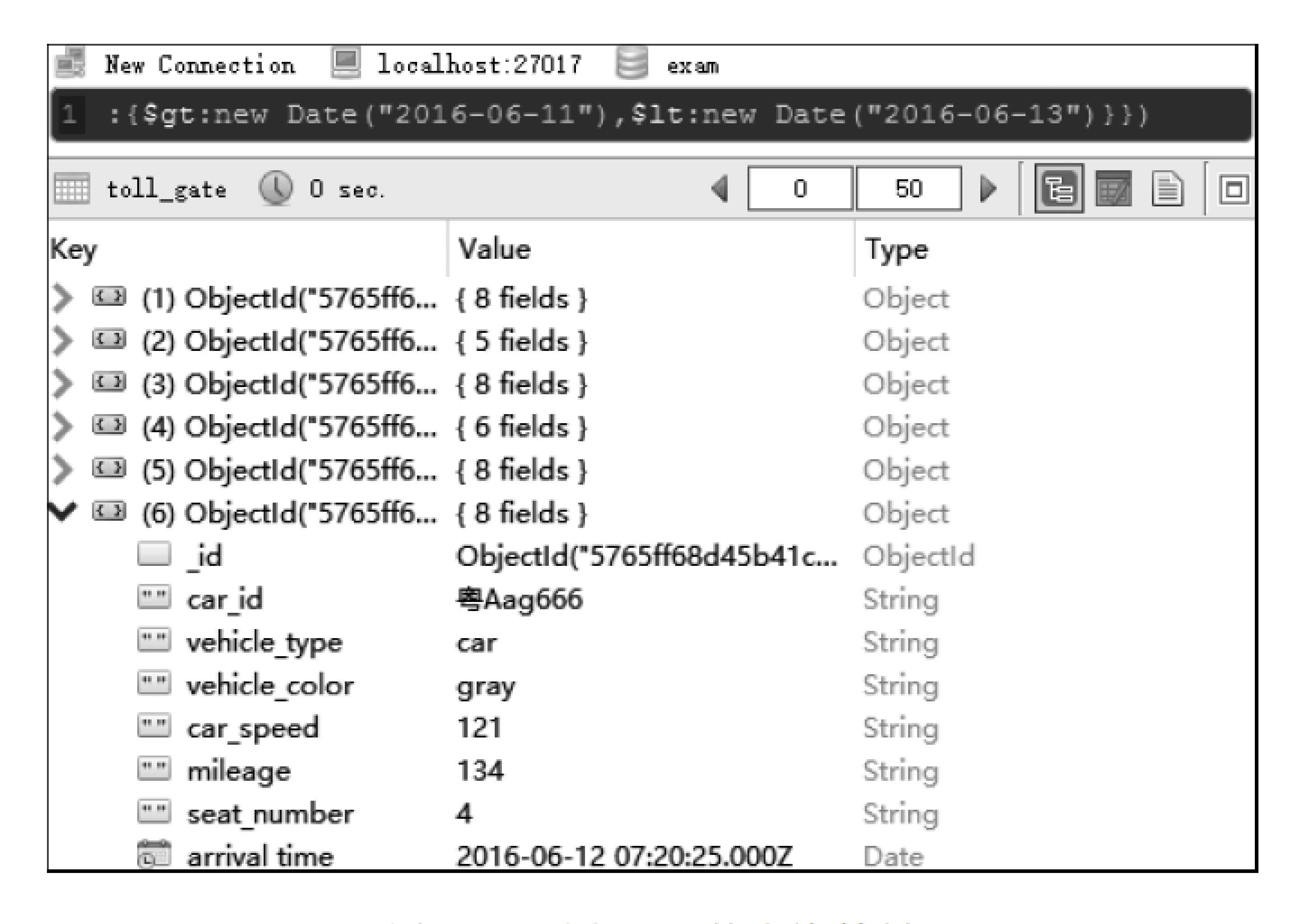
图 9-23　成功添加 10 个文档



```
New Connection   localhost:27017   exam
1 :{$gt:new Date("2016-06-11"),$lt:new Date("2016-06-13")}})
```

图 9-24 查询 arrival time 的值为 2016-06-12 的文档

查询结果如图 9-25 所示。

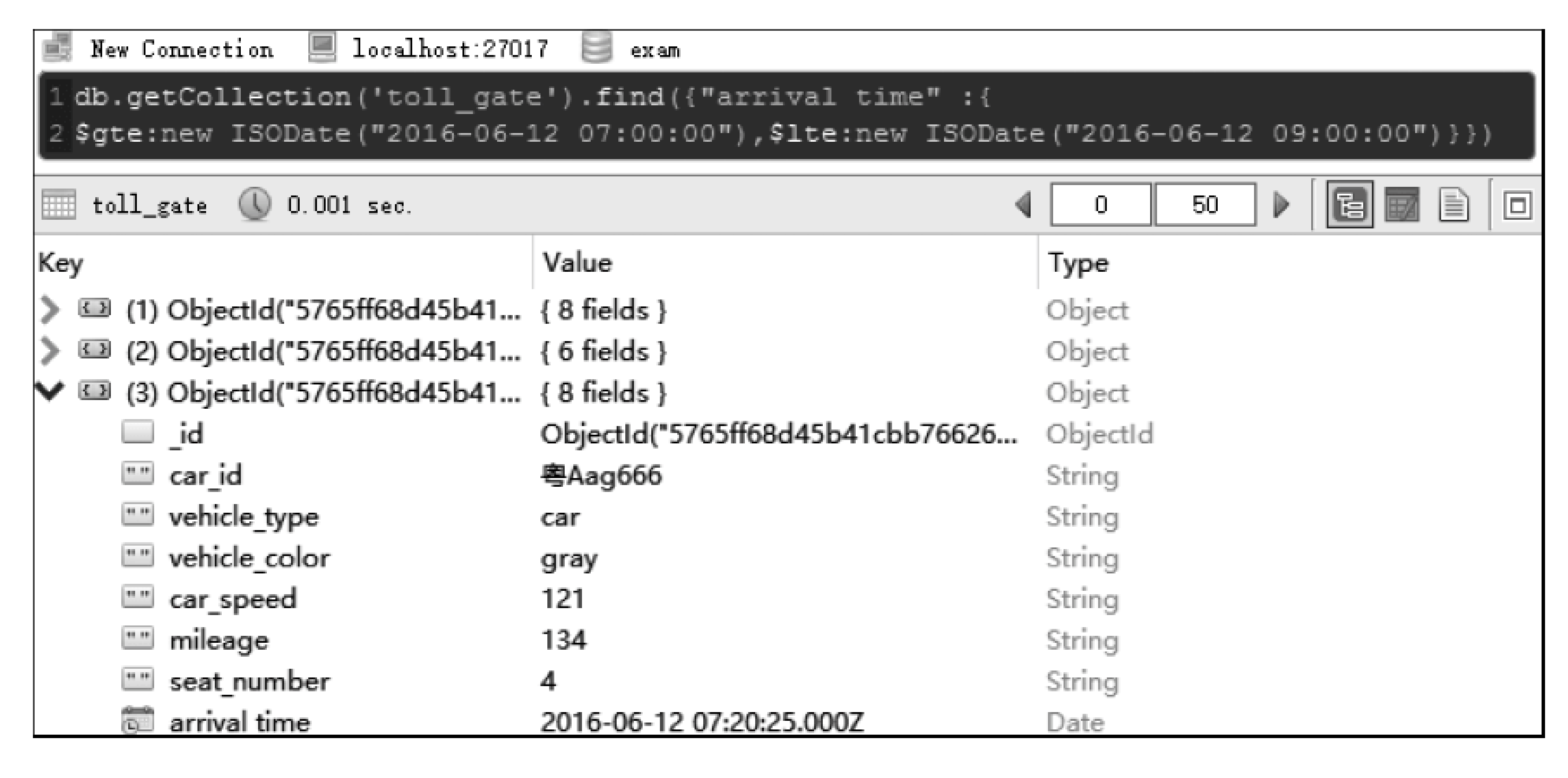


```
New Connection   localhost:27017   exam
1 :{$gt:new Date("2016-06-11"),$lt:new Date("2016-06-13")}})
toll_gate   0 sec.   0   50
```

Key	Value	Type
(1) ObjectId("5765ff6...	{ 8 fields }	Object
(2) ObjectId("5765ff6...	{ 5 fields }	Object
(3) ObjectId("5765ff6...	{ 8 fields }	Object
(4) ObjectId("5765ff6...	{ 6 fields }	Object
(5) ObjectId("5765ff6...	{ 8 fields }	Object
(6) ObjectId("5765ff6...	{ 8 fields }	Object
_id	ObjectId("5765ff68d45b41c...	ObjectId
car_id	粤Aag666	String
vehicle_type	car	String
vehicle_color	gray	String
car_speed	121	String
mileage	134	String
seat_number	4	String
arrival time	2016-06-12 07:20:25.000Z	Date

图 9-25 图 9-24 的查询结果

如需使用多条件查询，如查询 6 月 12 日 7 时整到 9 时整经过的车辆，输入命令及运行结果显示如图 9-26 所示。



```
New Connection   localhost:27017   exam
db.getCollection('toll_gate').find({"arrival time" :{
$gte:new ISODate("2016-06-12 07:00:00"),$lte:new ISODate("2016-06-12 09:00:00")}})
toll_gate   0.001 sec.   0   50
```

Key	Value	Type
(1) ObjectId("5765ff68d45b41...	{ 8 fields }	Object
(2) ObjectId("5765ff68d45b41...	{ 6 fields }	Object
(3) ObjectId("5765ff68d45b41...	{ 8 fields }	Object
_id	ObjectId("5765ff68d45b41cbb76626...	ObjectId
car_id	粤Aag666	String
vehicle_type	car	String
vehicle_color	gray	String
car_speed	121	String
mileage	134	String
seat_number	4	String
arrival time	2016-06-12 07:20:25.000Z	Date

图 9-26 多条件查询命令及查询结果

9.4 大数据与物联网和云计算

大数据、物联网和云计算都是在 21 世纪的第 2 个 10 年后得以迅猛发展并且形成了各自的技术领域。但三者无论是从产生的实际背景、技术的交叠借用和应用的相辅相成都有着密切的关联。

9.4.1 大数据与物联网

物联网是一种网络环境，通过射频识别、红外感应器、全球定位系统、激光扫描器和气体感应器等信息传感设备，按约定的协议将各类物品与互联网连接起来，彼此之间进行通信联络与信息交换，以实现智能化识别、定位、跟踪、监控和管理。物联网的核心和基础是互联网，是网络技术由"人-人"的互联到"人-物"和"物-物"互联的革命性拓展。物联网的终端延伸和扩展到了人与各类物品和物品与物品之间，也被称为继计算机、互联网之后信息产业发展的第三次浪潮。

物联网的突出特征是，处于环境中的每个物体(包括人)都可进行通信、寻址和控制，并且在未来将人们所涉及的任何物体实现上网联网。物联网增强了人们监控和测量真实世界中发生事情的能力，如发动机上的传感器传递了温度、速度和燃料损耗等数据，给予了人们精确了解设备实时工作状态的本领。

人们所涉及接触到的物品远远超过人本身的数量，因此物联网运行过程中每时每刻都会产生着体量巨大的数据和相应的数据管理业务。正是由于对相关数据和数据业务的不可思议的巨大需求，使得大数据与物联网的结合成为一种自然而然的结果。

物联网中产生的物联网大数据主要有下述两种情形。

(1) 物联网状态数据：这是最主要的物联网大数据。实际上所有终端设备都会产生类似的数据。状态数据可作为实时数据直接提供价值，也可作为原始数据进行更复杂的分析而创造新的价值。例如，停车场的车位监考设备，提供实时车位状态信息，能及时让车主了解空余车位的情况；发动机上采集的状态数据，与以往报废发动机状态数据进行相关性分析，可预判更新发动机的时机，减少损失。

(2) 物联网定位数据：实际上是 GPS 应用的必然结果。定位数据可广泛应用于公交车定位、物流信息反馈和服务跟踪等方面，也可服务科学研究，如给滇金丝猴戴上 GPS 项圈，记录活动轨迹，研究其生活习性。

将物联网感知的数据与其他移动互联网、常规互联网采集的数据结合，就形成了大数据重要的数据来源基础。

物联网可以汇聚大量数据，但还需要具有访问、建模和分析的能力。大数据助力物联网，大数据分析为物联网提供有用的分析，获取价值。大数据产生的原因主要归结于互联网、移动设备、物联网和云计算等快速崛起。

物联网发展离不开大数据，大数据推动物联网的发展。新时代对"智慧"的要求，外在表现就是物联网，而技术内涵之一就是大数据。

9.4.2 大数据与云计算

作为一种新的大规模分布式计算模式，云计算从云端按需获取所需要的服务。云计算本身也是一种数据处理技术，从某种意义上可以看作是大数据的一种业务模式。随着数据体量的急速增加，如何高效地获取数据，有效地深加工并最终形成有价值的信息，以及用更经济的方式存储结构复杂的大量数据等都是人们面临的挑战性课题，这些都需要"云"来提供存储、访问和计算的各类强有力的保障性服务。

大数据的价值在于对巨量数据进行基于全样本和分布式的数据挖掘，其特点是采用的

数据越来越多,需要进行的计算量越来越大、数据管理越来越呈现出动态化的形势、数据处理要求越来越实时,使得它必须依托云计算的分布式处理、分布式数据库、云存储和虚拟化技术。云计算也可以看作是在大数据背景下催生出来的一种基础架构和商业模式。云计算技术将分布各地的 CPU 整合管理使用,具备动态资源分配和调度、虚拟化和高可用性的特点,有效地降低成本,提高数据挖掘效率。

从技术上看,云计算关键技术中所涉及的海量数据存储技术、海量数据管理技术和 MapReduce 编程模型,都是大数据技术的基础。

云计算和大数据的关系是静与动的关系。云计算强调的是计算,即处理数据的行动,这就是“动”的概念;大数据则是实施计算的对象,一旦形成就相对稳定,这就是“静”的含义。结合实际应用,前者更强调计算处理能力,后者却看重存储管理能力。如果将大数据看作是一笔巨大财富,其中蕴含着极具价值的宝藏,云计算就是挖掘和利用宝藏的利器。缺乏强大的计算能力,数据宝藏终究是云中之月;没有大数据的积累积淀,云计算也只能是屠龙之刀。

9.4.3 大数据、物联网与云计算

有人提出过“互联网的未来功能和结构将与人类大脑高度相似,也将具备互联网虚拟感觉,虚拟运动,虚拟中枢和虚拟记忆神经系统”的说法。物联网对应互联网的感觉认知和运动神经系统,是“互联网大脑”收集信息的来源,就像人类的眼、耳、口、鼻和四肢等感知器官,源源不断地向互联网大数据汇聚数据和接收数据。云计算则对应于中枢系统,也就是相当于人的“大脑”。作为互联网的关键硬件层和核心软件层的集合,云计算应当是互联网智慧和意识产生的基础。大数据代表了互联网的数据信息层,而物联网、传统互联网及移动互联网都在源源不断地向互联网信息层汇聚数据和接收数据。可将物联网、云计算与大数据三者关系表现如图 9-27 所示。

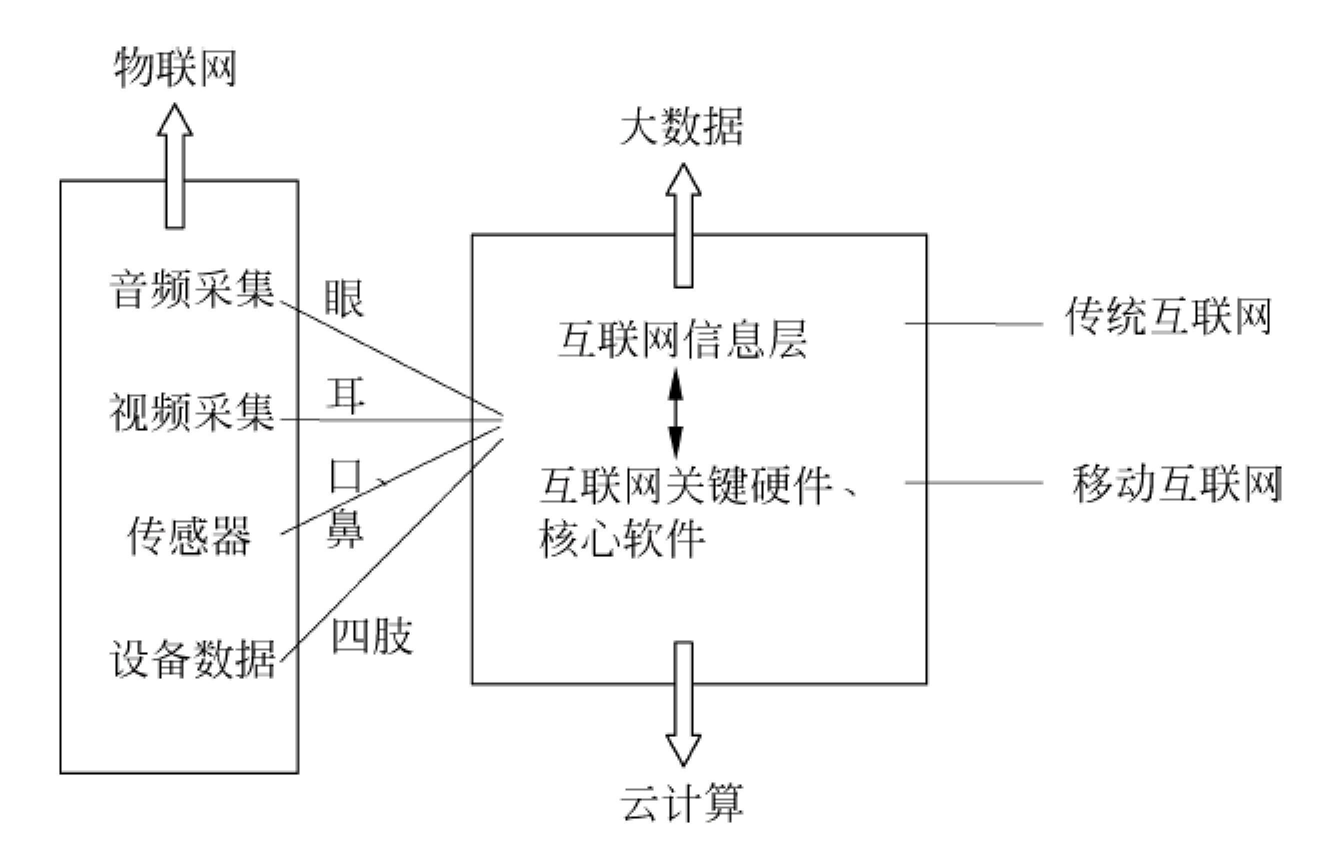


图 9-27 物联网、云计算与大数据三者关系

感应识别、网络传输、管理服务和综合应用是物联网的 4 个基本组成部分,其中网络传输和管理服务都需要使用云计算技术,因为使用云计算可能是物联网流通过程中的一种更为经济的方式。

首先,建设物联网除了需要传感器和传输通道外,还需要高效的、动态的和可以大规模

扩展的技术资源处理能力，云计算带来的高效率的运算模式正好可以为其提供良好的应用基础。云计算是实现物联网的核心，运用云计算模式使物联网中以兆计算的各类物品的实时动态管理和智能分析变得可能。

其次，云计算促进物联网和互联网的智能融合，从而构建智慧地球。物联网和互联网的融合，需要更高层次的整合，同样也需要依靠高效的、动态的、可以大规模扩展的技术资源处理能力，而这正是云计算模式所擅长的。

最后，物联网的发展又推动了云计算技术的进步，因为只有真正与物联网结合后，云计算才算是真正意义上从概念走向应用，两者缺一不可。

大数据、物联网和云计算互生互存和共欣共荣，共同推动信息技术向前迅猛发展。

本章小结

大数据是来自网络空间的一类特定数据集合，自其出现以来，得到了多方面的关注与重视。其影响甚至超过了 20 世纪计算机的诞生发展，也超过了 21 世纪互联网的迅猛普及。

对于政府来说，大数据关系到国家竞争力、关系到国家发展、关系到国民大众，由此世界主要强国都竭力推行大数据战略。

对于商界而言，大数据已经进入实用，其关联的商业价值重大，是企业竞争的利器坚具，由此跨国公司巨头都已率先投身于大数据开发与使用。

基于计算机科技本身考虑，大数据技术带来了新的挑战，带来了数据管理方面学科整合与技术攻关的重要机遇，由此诞生了一个极具发展前景的学科方向——数据科学。

大数据是一个诞生不久并正在迅速发展的新的数据管理技术领域，对于大数据概念的科学含义仍在探讨过程中，这实际上也正是大数据学科具有旺盛生命力和广阔发展前景的基本标志。大数据概念在技术层面上的内涵主要包含在 4V 特征描述中，应用层面上的内涵主要就是着眼于决策问题。其语义表示中的“大”或者说“多”不仅具有其字面上常规含义，还应该具有与一般常说的“数据”或“海量数据”的差异。大数据首先是需要其体量大到 PB 级别，量变引发质变，这个级别的数据量是常规数据管理技术所难以胜任的，随之会带来一系列新的技术甚至是学科方面的问题。PB 量级的数据难以使用常规方法进行采集；PB 量级的数据不会只有单一的数据来源，由此带来相异的数据类型和多样的数据格式；PB 量级的数据还具有数据产生与数据消耗的时间窗口严格限定，以及数据价值密度低的特点。所有这些，自然就带来了数据存储和数据分析的挑战：PB 量级的数据只能进行分布式存储；分布式存储的数据需要配置并行或并发计算模式；巨大的数据量和广泛的分布范围通常都不能完全基于专用的服务器网络系统，在实际应用中需要调用成千上万廉价服务器和 PC 以组成对于用户透明的大数据存储与处理网络等。由此也就形成了一系列技术架构、技术研发甚至学科级别的重要研究课题。

无论大数据还是常规数据都需遵循数据处理的一般规律，大数据处理的基本流程与常规数据本质上并无太大差异，区别在于相应技术框架与技术实现方面。由于要处理巨量的非结构化数据，当前大数据在各个处理环节中采用不同于常规数据并行处理（MPI）的 MapReduce 方式，而数据存储管理方面则采取数据结构要求更加宽松的 NoSQL 数据库。

大数据管理是对现有数据库技术的挑战，其带来的影响是多方面的。例如，大数据时代

处理数据理念上的三大转变：由抽样分析到整体分析转变，由精确分析到容错分析转变，由绞尽脑汁地“知其所以然”到只关注价值实现的“知其然”的转变等。当然，这并不是对“抽样”“精确”和“因果”分析思维的否定，而是在更广阔的视野下讨论了这些常规和经典分析模式的边界。不同的情形需要采用不同的应对处理，这不仅只是针对具体的技术手段，也可以针对更高层面的思维模式而言。事实一再表明“上帝的归上帝，凯撒的归凯撒”这一至理名言确实具有一般的更为广泛的意义。

技术革新特别是技术创新对于人类世界可以产生巨大的和多方面的影响，较远的如20世纪产生发展的计算机技术，较近的如21世纪来势迅猛的互联网技术。对于大数据技术，不少人也怀有类似的期望。现在看来，万事万物的数字数据化进程和PB量级数据交叉复用带来的巨大价值显现有可能为人类社会塑建起所谓的数据文化氛围与环境。

当然，现今大数据技术本身也面临着产业生态环境、数据安全隐私和信息公正公开等各类挑战性课题。正如大数据概念的学科内涵与技术外延还在探讨之中那样，这些都是来自于大数据旺盛生命力的内在驱动。

大数据时代中大数据与云计算、物联网的关系相辅相成，密不可分。大数据产生的原因离不开物联网等提供的巨量数据，大数据分析则使之转换为价值，又离不开云计算对海量数据的分布式存储、分布式管理的能力。总之，物联网与云计算助力大数据革命，共同推动了大数据时代的到来。

主要参考文献

[1] 迈尔-舍恩伯格库克耶.大数据时代[M].盛杨燕，周涛，译.杭州：浙江人民出版社，2013.
[2] 朱扬勇，熊赟.大数据是数据技术还是应用[J].大数据，2015，1(1)：71-81.
[3] 涂子沛.大数据：正在到来的数据革命(3.0升级版)[M].桂林：广西师范大学出版社，2015.
[4] 伊恩·艾瑞斯. 大数据思维与决策[M].宫相真，译.北京：人民邮电出版社，2014.
[5] 林子雨.大数据技术原理与应用[M].北京：人民邮电出版社，2016.
[6] 佐佐木达也，著.NoSQL数据库入门[M].罗勇，译.北京：人民邮电出版社，2012.
[7] 郭远威.大数据存储MongoDB实战指南[M].北京：人民邮电出版社，2015.
[8] 娄岩.大数据技术概论[M].北京：清华大学出版社，2017.

第 10 章 时态数据索引技术

网络数据和移动对象等新型数据的数据量巨大、数据结构复杂、查询类型众多和应用范围广泛，使用相应的数据库技术对其进行有效管理是进入 21 世纪后对计算机科学技术应用的重要挑战之一。网络数据中的 XML 和移动对象数据都可以建立起合适的数据模型，因此建立相应的 XML 数据库和移动对象数据库就有了必需的基础。实际上，它们已经成为了当前数据库技术的热点领域之一。如前所述，从事务管理进入到时间和空间维度考量是数据管理实际应用的驱动，也是数据库技术深入发展的必须。添加了时间标签的时态 XML 数据是常规 XML 数据的重要组成部分，而移动对象数据本身就与时间与空间因素密切相关。由于难以建立像 RDB 那样成熟的商业化 DBMS，而数据管理核心是数据查询，因此，面对实际应用的强大推动，近年来面向两者的数据索引查询技术日益引起人们的关注，成为研究新型数据管理的一项基本工作。本章讨论一种时态数据索引技术（TDindex），并以此为基础，着眼于时间查询与相应特定数据类型自身特征查询的协同整合，分别讨论时态 XML 数据索引（TX-tree）和移动对象数据索引（pm-tree）。

10.1 时态数据索引概述

基于有效时间的时态数据（Temporal Data）可以表示为二元组 Td =< D，VT >，其中 D 是常规的不带时间标签的数据即非时态数据。设 VTs≤VTe，以 VTs 和 VTe 为始点和终点的有效时间期间 VT 标记为 VT=[VTs，VTe)，并将 Td 的有效时间期间记为 VT(Td)。

从数学角度考虑，VT=[VTs，VTe)可看作 VTs-VTe 平面上的点（VTs，VTe）。如果将给定时态数据集合 E 的相同的时间期间只计算一次并将其记为 Γ，由此就建立起 Γ 到 VTs-VTe 平面一个点集 $H(\Gamma)$之间的一一对应关系 $\Gamma \leftrightarrow H(\Gamma)$：

$$VT = [VTs, VTe) \rightarrow P = (VTs, VTe)$$

本章讨论中，在不引起混淆情况下将不刻意区分 Γ 和 $H(\Gamma)$。

$\forall P_0 \in H(\Gamma)$，$col(P_0)$表示 $H(\Gamma)$中满足条件 $VTs(P)=VTs(P_0)$的点 P 的集合，即

$$col(P_0) = \{P \mid (VTs(P) = VTs(P_0)) \land P \in H(\Gamma)\}$$

10.1.1 基于拟序时态数据结构

数据索引实际上都是以给定数据集合上的某种数据结构为基础的。以下讨论基于拟序关系的时态数据结构。

1. 时态拟序关系

为构建所需的时态数据结构，引入下述相关概念。

【定义 10-1】 (拟序关系和线序分枝)设 E 是时态数据集合。

(1) E 上的时态关系$\lesssim$。对于 Td_1、$\mathrm{Td}_2 \in E$,定义 E 上的时态关系$\lesssim$如下。

$$\mathrm{Td}_1 \lesssim \mathrm{Td}_2 \Leftrightarrow \mathrm{VT}(\mathrm{Td}_1) \subseteq \mathrm{VT}(\mathrm{Td}_2)$$

当$\neg(\mathrm{Td}_1 \lesssim \mathrm{Td}_2) \wedge \neg(\mathrm{Td}_2 \lesssim \mathrm{Td}_1)$,则称 Td_1 和 Td_2 时态不相容,记为 $\mathrm{Td}_1 \lesssim\gtrsim \mathrm{Td}_2$。

(2) E 上的拟序关系。如果 R 是集合 E 上一个满足自反和传递的关系,则称 R 是 E 上的一个拟序(quasi-order)。

可以验证,(1)中$\lesssim$为拟序关系,称其为时态数据集合 E 上的时态拟序(temporal quasi-order)。

(3) 线序分枝。设 E 是具$\lesssim$的时态拟序集合,E 中一个全序分枝称为 E 的一个线序分枝(Linear Order Branch, LOB)。

由于主要讨论数据的时态操作,为叙述简洁,本章中除特殊说明外,时态数据集 E 上拟序也看作是时间期间集合 Γ 上的拟序。

以下讨论中通常将 Td 和 VT(Td)“有意混用”地记为 u、v 和 w 等。

2. 下右优先算法(DRFT)

时间期间集合 Γ 上元素按照拟序关系组织成为线序分枝的集合实际上就是建立起 Γ 上的数据结构,为此首先需要讨论 Γ 中元素基于拟序的遍历算法。

算法基本思想:从 $H(\Gamma)$“最左上方”点开始,根据同列优先原则进行构建 LOB,直到 $H(\Gamma)$所有元素被选入相应 LOB 终止。在 LOB 中的元素满足前一个元素的时间期间包含后一个元素的时间期间。这就是下述的下右优先遍历算法(DRFT)。

1) $H(\Gamma)$“下(右)优先”遍历

【算法 10-1】 下右优先遍历算法(Down and Right First Traverse, DRFT)

由 $H(\Gamma)$“最左上方”点 $u(i,j)$开始,$P=u(i,j)$,$L_k=\{P\}$,$k=1$。

(1) 沿 $\mathrm{col}(i)$遍历到 $H(\Gamma)$“最后”点 $K(i,m)$,$i \leqslant m \leqslant j$。将遍历到的点依次放入 L_k,$P=K(i,m)$。

(2) 考察是否存在 $N(i+1,m) \in H(\Gamma)$,满足 $VTe(N) \leqslant VTe(K)$,若存在,将其放入 L_k,返回步骤(1);否则,$i{+}{+}$,检查是否 $i=m$,若是,执行(3),若否,继续执行(2)。

(3) 输出列表 L_k。

(4) $H(\Gamma)=H(\Gamma) \setminus L_k$,若 $H(\Gamma) \neq \varnothing$,$k{+}{+}$,返回步骤(1)。

按 DRFT 算法得到线序分枝 LOB=$<P_1,P_2,\cdots,P_k>$和 $H(\Gamma)$遍历序列$\sum(\Gamma)=<L_1,L_2,\cdots,L_m>$,其中 $L_i(1 \leqslant i \leqslant m)$是遍历子集都按照算法获得的次序排序。

设 $\mathrm{VTs}(\Gamma)=\max\{\mathrm{VTs}(u) \mid u \in \Gamma\}$,$\mathrm{VTe}(\Gamma)=\max\{\mathrm{VTe}(u) \mid u \in \Gamma\}$,则算法 10-1 时间复杂度为 $O((\mathrm{VTs}(\Gamma) \times \mathrm{VTe}(\Gamma))/2)$。

2) $\sum(\Gamma)$基本性质

由 DRFT 算法获得的遍历分枝序列 $\sum(\Gamma)$ 性质定理。

【定理 10-1】 (遍历分枝序列 $\sum(\Gamma)$ 性质)下述结论成立。

(1) $\sum(\Gamma)=<L_1,L_2,\cdots,L_m>$是 Γ 的一个划分(partition)。

(2) 设 $\mathrm{VTs}(L_i)$和 $\mathrm{VTe}(L_i)$分别是 $L_i(1 \leqslant i \leqslant m)$中元素始点和终点序列,$\mathrm{VTs}(L_i)$单调增加,$\mathrm{VTe}(L_i)$单调减少。

由 DRFT 算法就可以得到定理的证明。

按照定义 10-1,由算法 10-1 得到的 Γ 遍历子集 L_i 就是 Γ 的 $\mathrm{LOB}_i(1\leqslant i\leqslant m)$。上述定理说明,当完成拟序遍历之后,对于每个线序分枝 LOB 来说,还得到了两个进行了排序的序列,即单调递增始点序列和单调递减的终点序列,这对于此后建立索引的查询算法是非常重要的。

3. 线序划分与最小线序划分

前述对时间期间集合 E 进行拟序遍历结果就是将 E 中的数据进行分类,每个 LOB 就表示一种类型,类型中的元素具有"一个包含一个"的全序结构。由于构建 DRFT 的特定要求,得到的 LOB 都是彼此不交的。在离散数学中,如果一个集合 E 被分隔为多个子集的并集,并且这些子集彼此不交,所有这样子集的集合就构成了 E 上的一个划分。因此由 *DRFT* 得到的全体线序分枝集合就构成了 E 上的一个划分。

1) 线序划分与数据结构

【定义 10-2】 (线序划分)由 DRFT 算法得到的 $\sum(\Gamma)=<L_1,L_2,\cdots,L_m>$。$\forall L_i$,$L_j\in\sum$,$i\neq j$,$L_i\cap L_j=\varnothing$,且 $\cup L=\Gamma\ (1\leqslant i,j\leqslant m)$,定义为 Γ 上的一个线序划分(Linear Order Partition, LOP)并记为 $\mathrm{LOP}(\Gamma)=<L_1,L_2,\cdots,L_m>$。

由 $\sum(\Gamma)$ 定义了 Γ 上的拟序关系时态数据结构(Quasi-Order Temporal Data Structure,QOTDS)。

设 $u_0\in\mathrm{LOB}$, LOB 包括 u_0 在内的所有 u_0 的"前驱"构成的片段记为 $\mathrm{Lp}(u_0)$,包含 u_0 在内的所有 u_0"后继"元素构成的片段记为 $\mathrm{Ls}(u_0)$。

2) 四分区域及其性质

可以通过 $H(\Gamma)$中给定点 u_0 将整个划分为与 u_0 具有拟序关联的 4 个区域。

【定义 10-3】 ($H(\Gamma)$四分区域)$\forall u_0\in H(\Gamma)$,则 u_0 将 $H(\Gamma)$分为如下 4 个子区域。

(1) $\mathrm{LU}(u_0)=\{u\mid u\in H(\Gamma)\wedge \mathrm{VTs}(u)\leqslant\mathrm{VTs}(u_0)\wedge\mathrm{VTe}(u_0)\leqslant\mathrm{VTe}(u)\}$。

(2) $\mathrm{LD}(u_0)=\{u\mid u\in H(\Gamma)\wedge \mathrm{VTs}(u)<\mathrm{VTs}(u_0)\wedge\mathrm{VTe}(u)<\mathrm{VTe}(u_0)\}$。

(3) $\mathrm{RU}(u_0)=\{u\mid u\in H(\Gamma)\wedge \mathrm{VTs}(u_0)<\mathrm{VTs}(u)\wedge\mathrm{VTe}(u_0)<\mathrm{VTe}(u)\}$。

(4) $\mathrm{RD}(u_0)=\{u\mid u\in H(\Gamma)\wedge \mathrm{VTs}(u_0)\leqslant\mathrm{VTs}(u)\wedge\mathrm{VTe}(u)\leqslant\mathrm{VTe}(u_0)\}$。

$\mathrm{LU}(u_0)$、$\mathrm{LD}(u_0)$、$\mathrm{RU}(u_0)$和 $\mathrm{RD}(u_0)$分别称为 $H(\Gamma)$关于 u_0 的"左上""左下""右上"和"右下"子区域。为叙述方便,定义 $\mathrm{RD}(u_0)$中子集 RDO 如下。

$$\mathrm{RDO}(u_0)=\{u\mid u\in H(\Gamma)\wedge(\mathrm{VTs}(u_0)\leqslant\mathrm{VTs}(u))\wedge(\mathrm{VTe}(u)<\mathrm{VTe}(u_0))\}$$

【例 10-1】 基于 $u_0=35$ 的 $H(\Gamma)$四分区域划分如图 10-1 所示。

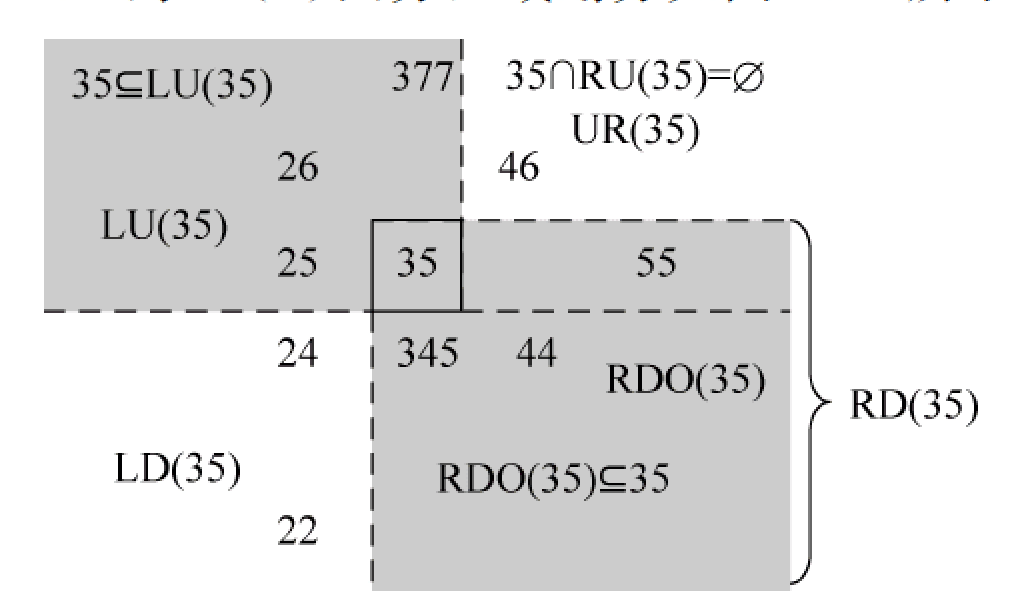


图 10-1 基于 $u_0=35$ 的 $H(\Gamma)$四分区域划分

【定理 10-2】 (四分区域性质定理)$\forall u_0 \in H(\Gamma)$,下述结论成立:

(1) $u_0 \lesssim v_0 \Leftrightarrow v_0 \in \mathrm{LU}(u_0)$。

(2) $v_0 \lesssim u_0 \Leftrightarrow v_0 \in \mathrm{RDO}(u_0)$。

(3) $u_0 \lessgtr v_0 \Leftrightarrow v_0 \in \mathrm{RU}(u_0) \vee v_0 \in \mathrm{LD}(u_0)$。

(4) $v_0 \in \mathrm{RU}(u_0) \Rightarrow u_0 \cap v_0 = \varnothing$。

证明(1):设 $u_0 = [i_0, j_0)$,$v_0 = [k_0, l_0)$,$u_0 \lesssim v_0 \Leftrightarrow u_0 \subseteq v_0 \Leftrightarrow k_0 \leqslant i_0, j_0 \leqslant l_0 \Leftrightarrow v_0 \in \mathrm{LU}(u_0)$,同理可证(2)、(3)和(4)。

3) 最小线序划分

如同对树和图进行某种意义下的遍历具有多种方式,在时间期间集合 E 上实施拟序遍历还可以有多种方式,并由此建立起相应的线序分枝 LOB 的集合,进而还可以认为由此构建的线序划分 LOP 也可有多种方式,由于线序划分是建立相应时态索引的基本出发点,从数据操作效率考虑,选用的基于 LOP 的时态数据结构应当尽可能简洁。具体而言,就是采用的 LOP 应当具有较少 LOB 个数,或者等价地讲,LOP 中的 LOB 应当包含尽可能多的时间期间。因此,对于 E 中可能的 LOP,可以将 LOP 中包含多少的 LOB 作为 LOP“优劣”的一种判定标准。

【定义 10-4】 (最小线序划分)设 LOP_0 是 Γ 上线序划分,对任意 Γ 上的 LOP,如成立 $|\mathrm{LOP}_0| \leqslant |\mathrm{LOP}|$,则称 LOP_0 是 Γ 上最小线序划分(Minimum Linear Order Partition, MLOP)。

由上述引入的四分区域概念,可以证明 MLOP 的存在性定理。

【定理 10-3】 (MLOP 存在定理)由算法 10-1 构建的 LOP 是 MLOP。

证明:设由算法 10-1 得到 $\mathrm{LOP} = < L_1, L_2, \cdots, L_m >$,其中 L_i 由计算顺序排序,则 $\forall u_{i0} \in L_i (1 < i \leqslant m)$,$\exists u_{k0} \in L_{i-1}$,$u_{i0} \lessgtr u_{k0}$。事实上,只需说明 $\exists v_0 \in L_{i-1}$ 且 $v_0 \in \mathrm{LD}(u_i 0)$。假设这样的 v_0 不存在,$L_{i-1} \subseteq \mathrm{LU}(u_i 0)$,则 $L_i \subseteq \mathrm{RD}(\min L_{i-1})$,与 LOB 是划分矛盾。设由此得到的结点为 $u_1, u_2, , \cdots, u_{m-1}$;设 u_m 是 L_m 上任意一个给定点,则成立 $u_{i-1} \in \mathrm{LD}(u_i)(1 < i \leqslant m)$,即 $u_1, u_2, \cdots, u_{m-1}, u_m$ 两两互不相容。任何线序划分至少有 m 个 LOB。定理得证。

定理 10-3 表明,按照 DRFT 算法得到的就是一种 MLOP,在上述意义下具有“最优性”。以下提及的 LOP 均指的是 MLOP。

10.1.2 时态数据索引

时态数据处理的关键技术是其中“时间元素”与“数据本体”之间的整合。具体整合实现方式由所处理时态数据的应用场景确定。

(1) 对于时态关系数据或时态对象关系数据来说,时间元素可以“逻辑”地看作是时态数据元组所具有多种“属性”中的一种(即时间属性),查询中可以采用先进行“时间”属性筛选再进行数据本体处理的“简单耦合”方式。

(2) 对于各类新型时态数据,如时态 XML 和移动对象数据等,应用场景更为复杂,时间元素与数据本体在处理过程中交织纠缠,需要在更为精细的场景层面上考虑“时间”与“非时间”的“协同整合”。

(3) 对于“简单耦合”,相应时态处理本质上可看作只是对“时间标签”的特殊处理。在数据处理过程中,“简单耦合”可借鉴经典的索引方法,如 B^+-tree。

(4) 对于“协同整合”，除对时间元素进行技术考量之外，还需要重点研究时间因素与应用场景的有效配置，即“协同整合”需要面临各类新的问题。可以认为，“协同整合”过程从索引构建角度来看就是相关时态数据的划分或分割。

前述的 LOP 实际上就是以“时间元素”为导向建立起来的一般时态数据结构。以其为基础，可建立处理“简单耦合”的索引框架 TQOindex 和处理“协同耦合”的索引框架 TDindex，前者是平衡树索引，后者是不具平衡性特征的一般树索引，可参见本章所附相关文献。本章主要研究 TDindex 的构建与应用。

1. TDindex 构建

对 $L\in \mathrm{LOP}(\Gamma)$，记 L 的首结点为 $\max(L)$，尾结点为 $\min(L)$。

(1) 定义 $\Gamma\max$ 为 $\mathrm{LOP}(\Gamma)$ 中所有“$\max(L)$”的集合，通过算法 10-1，在 $\Gamma\max$ 上进行线序划分，得到的 LOB 序列记为 $\mathrm{LOP}(\Gamma\max)=\{L_i(\Gamma\max)\}(1\leqslant i\leqslant|\mathrm{LOP}(\Gamma)|)$。

(2) 定义 $\Gamma\min$ 为 $\mathrm{LOP}(\Gamma)$ 中所有“$\min(L)$”的集合，通过算法 10-1，在 $\Gamma\min$ 上进行线序划分，得到的 LOB 序列记为 $\mathrm{LOP}(\Gamma\min)=\{L_r(\Gamma\min)\}(1\leqslant r\leqslant|\mathrm{LOP}(\Gamma)|)$。

(3) 定义 max(LOP) 为 LOP 中各个 LOB 的最大元组成的集合。

不至于混淆，可以将 $L_i(\Gamma\max)$ 和 $L_r(\Gamma\min)$ 看作“端点 LOB”，而将定义 10-1 中的线序分枝看作“数据 LOB”。

【定义 10-5】 (TDindex 结构)H(Γ)上 TDindex 是满足下述条件要求的四层树形结构：

(1) 叶结点层：即数据 LOB 层，本层结点为 $L(\Gamma\min(L_i(\Gamma\max)))$ 对应元素 LOB。

(2) 最小端 LOB 点层：即 LOP(Γmin)层，本层结点为 $L_i(\Gamma\min)$。

$$\mathrm{LOP}(\Gamma\min)=\{L_i(\Gamma\min)\}(1\leqslant i\leqslant|\mathrm{LOP}(\Gamma\min)|)$$

本层中每一结点 n_3 的子结点为叶结点层的相应结点，该结点的最小元包含在 n_3 中。

(3) 最大端点 LOB 层：即 LOP(Γmax)层，本层结点为 $L_i(\Gamma\max)$

$$\mathrm{LOP}(\Gamma\max)=\{L_i(\Gamma\max)\}(1\leqslant i\leqslant|\mathrm{LOP}(\Gamma\max)|)$$

本层中每一结点 n_2 的子结点为最小端点 LOB 层中满足条件的结点 n_3：如果 n_3 包含有叶结点层的结点 n_4 中的最小元，而作为数据 LOB 的结点 n_4 的最大端点位于 n_2 中。

(4) 根结点层：即 max(LOP(Γmax))层，其中唯一根结点由 max(LOP(Γmax))中的元素组成。

TDindex 时态索引结构如图 10-2 所示。

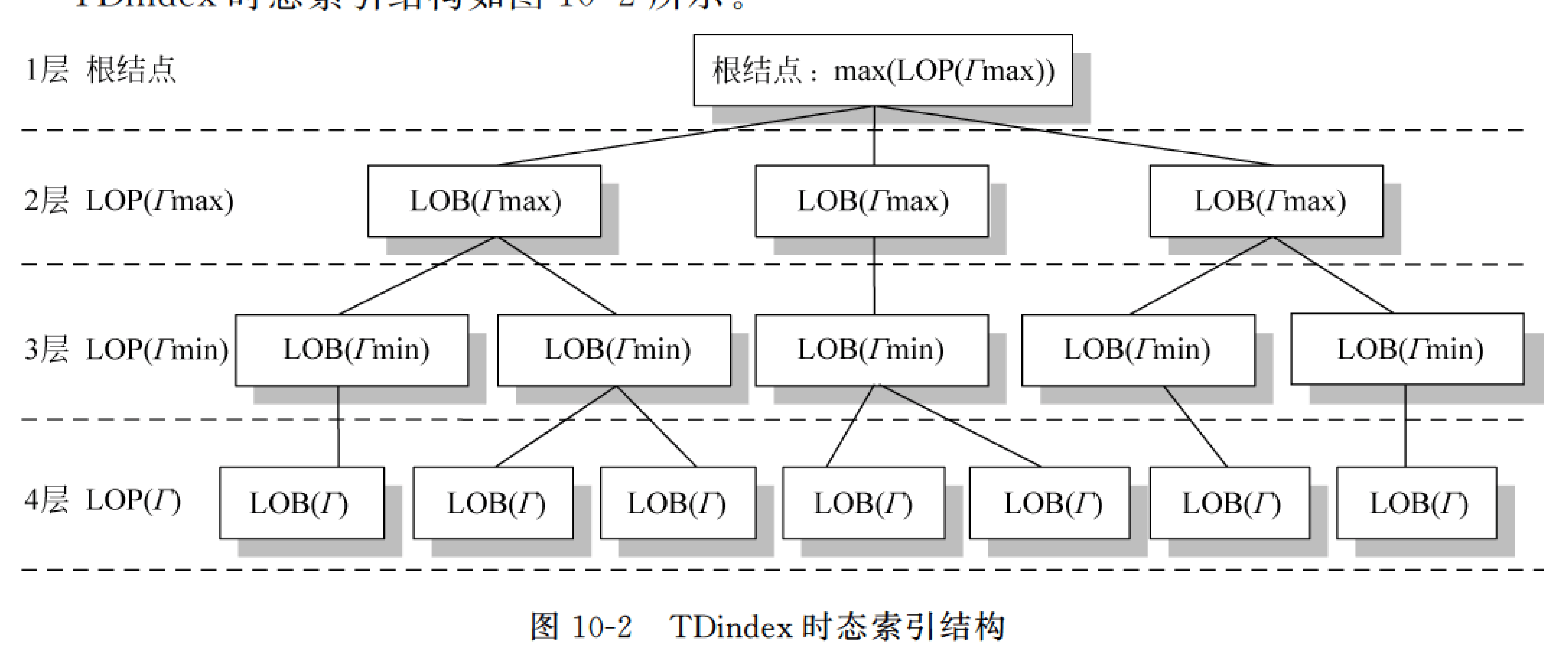


图 10-2 TDindex 时态索引结构

2. TDindex 示例

【例 10-2】 给定 Γ 上 LOP(Γ)=< L_1,L_2,L_3,L_4,L_5 >,如图 10-3 所示,其中

L_1 =<[0,9),[0,8),[0,7),[0,6),[1,6),[1,5),[1,4),[1,3),[2,3),[3,3) >。

L_2 =< [1,8),[1,7),[2,7),[2,4) >。

L_3 =< [2,8),[3,8),[3,6),[4,6),[4,5) >。

L_4 =< [4,9),[4,8),[4,7),[5,6) >。

L_5 =<[5,9),[6,8),[6,7) >。

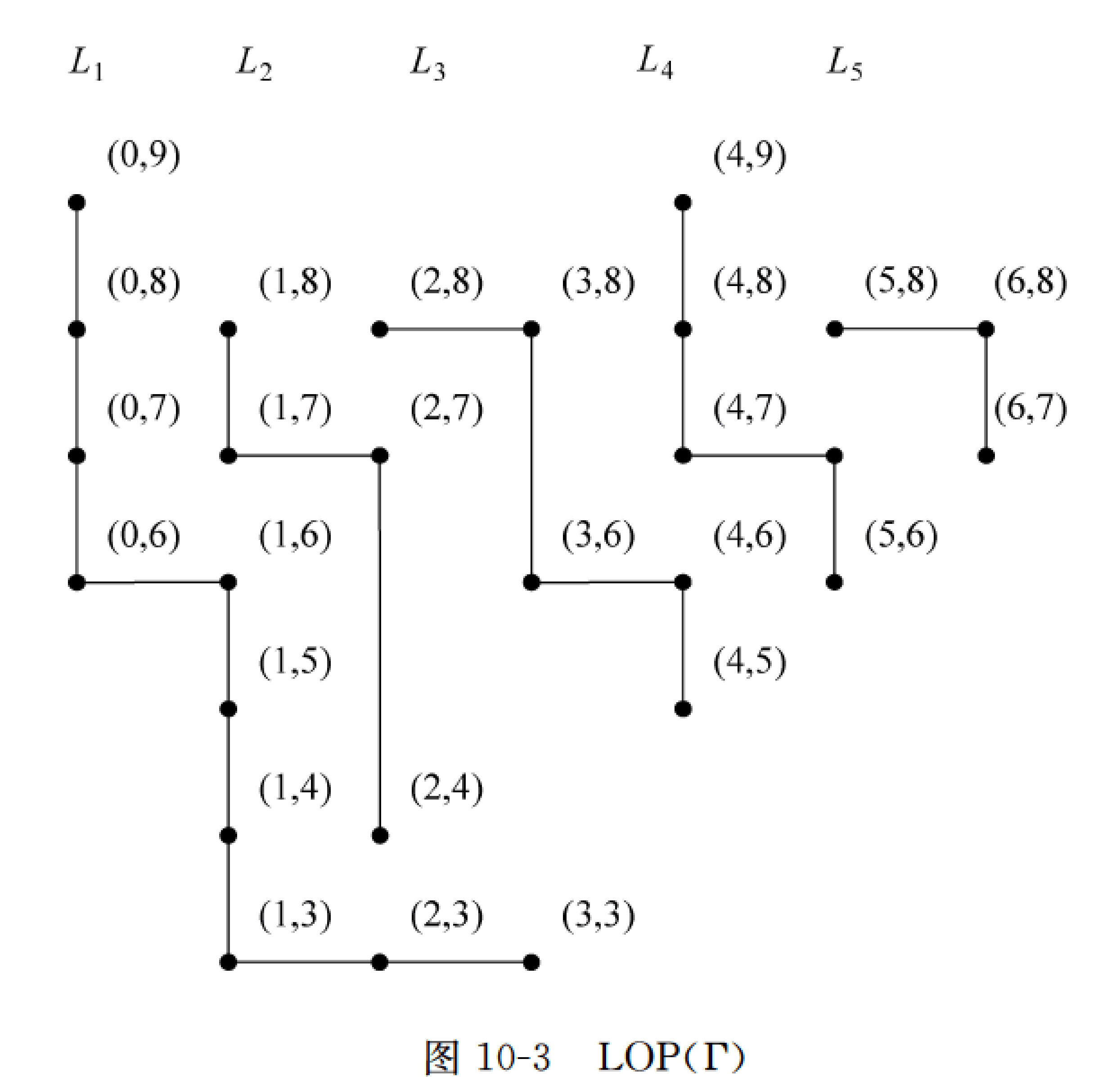


图 10-3 LOP(Γ)

Max(L_1)= [0,9),Max(L_2) = [1,8),Max(L_3) = [2,8),Max(L_4 = [4,9),)

Max(L_5) = [5,8),

Γmax= < Max(L_1), Max(L_2),Max(L_3),Max(L_4),Max(L_5) >

= <[0,9),[1,8),[2,8),[4,9),[5,8) >;

对 Γmax 由 DRFT 算法可得 LOP(Γmax)=< L_1(Γmax), L_2(Γmax)>,其中

L_1(Γmax) = {Max(L_1),Max(L_2),Max(L_3)}

= <[0,9),[1,8),[2,8), [5,8) >;

L_2(Γmax) = < Max(L_4) > = <[4,9) >;

max(LOP(Γmax)) = < max(L_1(Γmax)),max(L_2(Γmax)) >

= < [0,9),[4,9) >。

LOP(Γmax)如图 10-4 所示,其中实线表示对 Γmax 进行 DRFT 算法。

类似,由图 10-3 可知,Min(L_1)=[3,3),Min(L_2)=[2,4),Min(L_3)=[4,5),Min(L_4)=[5,6),Min(L_5)=[6,7).

Γmin= < Min(L_1), Min(L_2), Min(L_3),Min(L_4),Min(L_5) >

= <[3,3),[2,4),[4,5),[5,6),[6,7) >;

对 Γmin 由 DRFT 算法可得 LOP(Γmin)=< L_1(Γmin), L_2(Γmin)>,其中

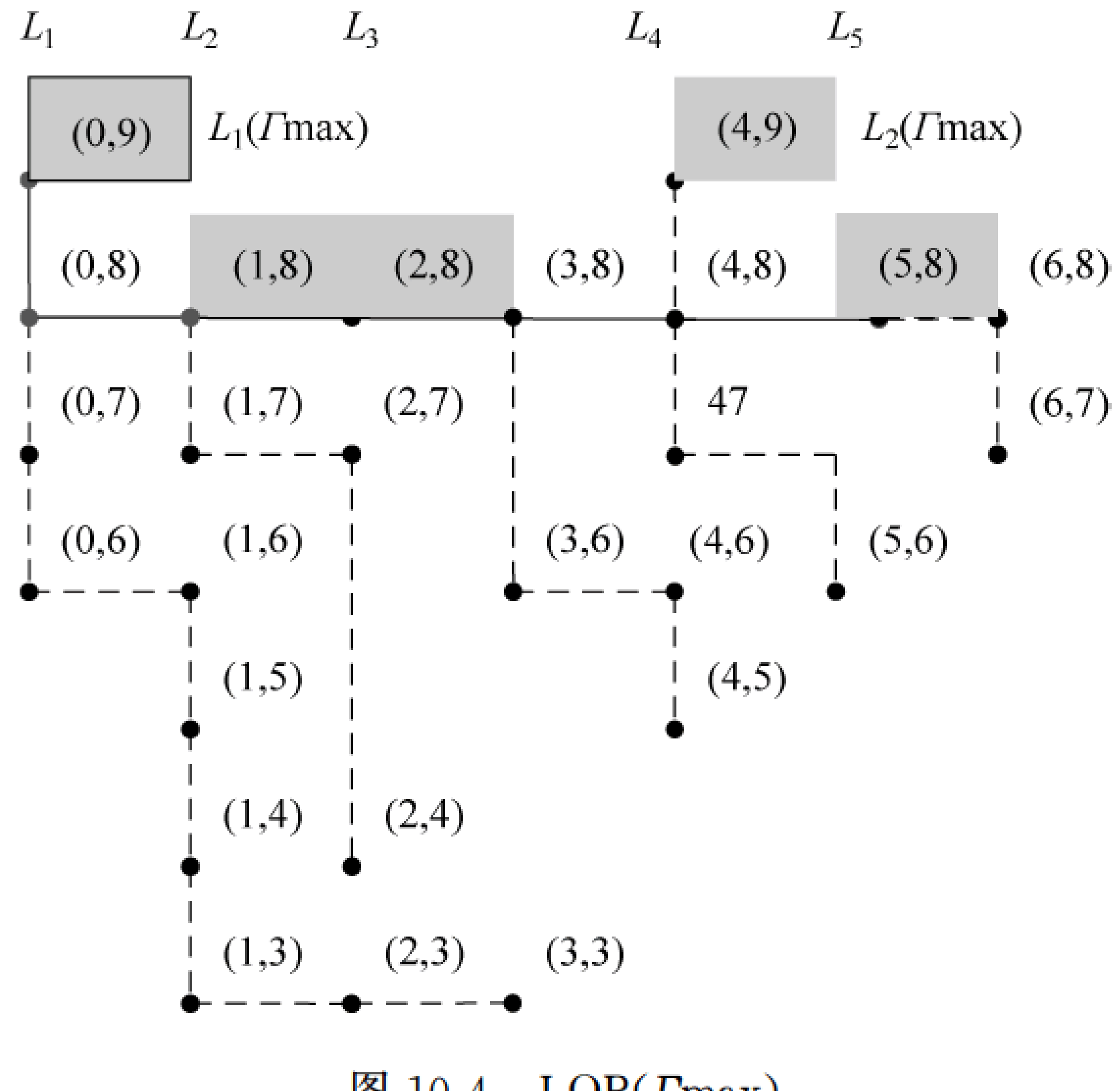


图 10-4　LOP(Γmax)

$$L_1(\Gamma\min) = \{\text{Min}(L_1), \text{Min}(L_2)\} = <[3,3),[2,4)>;$$
$$L_2(\Gamma\min) = <\text{Min}(L_3)> = <[4,5)>;$$
$$L_3(\Gamma\min) = <\text{Min}(L_4)> = <[5,6)>;$$
$$L_4(\Gamma\min) = <\text{Min}(L_5)> = <[6,7)>。$$

此时，相应 TDindex(Γ)索引实例中的结点构成如下。

(1) 叶结点层：由 5 个数据 LOB 构成 5 个结点，分别为 $L_1(\Gamma)$、$L_2(\Gamma)$、$L_3(\Gamma)$、$L_4(\Gamma)$、$L_5(\Gamma)$。

(2) 最小端点层：由 $L_1(\Gamma\min)$、$L_2(\Gamma\min)$、$L_3(\Gamma\min)$和 $L_4(\Gamma\min)$构成该层的 4 个结点，其中 $L_1(\Gamma\min)$的子结点为 $L_1(\Gamma)$和 $L_2(\Gamma)$；$L_2(\Gamma\min)$，$L_3(\Gamma\min)$和 $L_4(\Gamma\min)$分别以 $L_3(\Gamma)$、$L_4(\Gamma)$和 $L_5(\Gamma)$为各自的子结点。

(3) 最大端点层：$L_1(\Gamma\max)$和 $L_2(\Gamma\max)$构成本层的两个结点。由于 $L_1(\Gamma\min)$、$L_2(\Gamma\min)$和 $L_3(\Gamma\min)$的子结点 $L_1(\Gamma)$、$L_2(\Gamma)$、$L_3(\Gamma)$和 $L_4(\Gamma)$的最大元均在 $L_1(\Gamma\max)$，因此 $L_1(\Gamma\max)$有子结点 $L_1(\Gamma\min)$、$L_2(\Gamma\min)$和 $L_3(\Gamma\min)$；同时，$L_4(\Gamma\min)$子结点 $L_5(\Gamma)$的最大元在 $L_2(\Gamma\max)$中，即有 $L_2(\Gamma\max)$以 $L_4(\Gamma\min)$为子结点。

由此得到相应的时态数据索引 TDindex(Γ)如图 10-5 所示，其中图 10-5(a)表示实际数据情形，而图 10-5(b)表示带入相应符号后的情形。

10.1.3　TDindex 数据查询

时态查询是基于时间约束的查询，相关时间约束的谓词形式通常采用 Allen 的 13 种时态关系。为了描述简便清晰，本章时态查询中时间约束采用“包含约束”方式：设 Q 是给定的时态查询，TData 表示存储在数据库中相应的时态数据，需要查询所有满足 VT(Q)⊆VT(TData)的时态数据，其中 VT(.)表示相关时态数据的有效时间期间。

“包含”查询是一种基本的时态查询，其他如“相交”“相离”等查询都可以通过适当方式转换为“包含”查询。



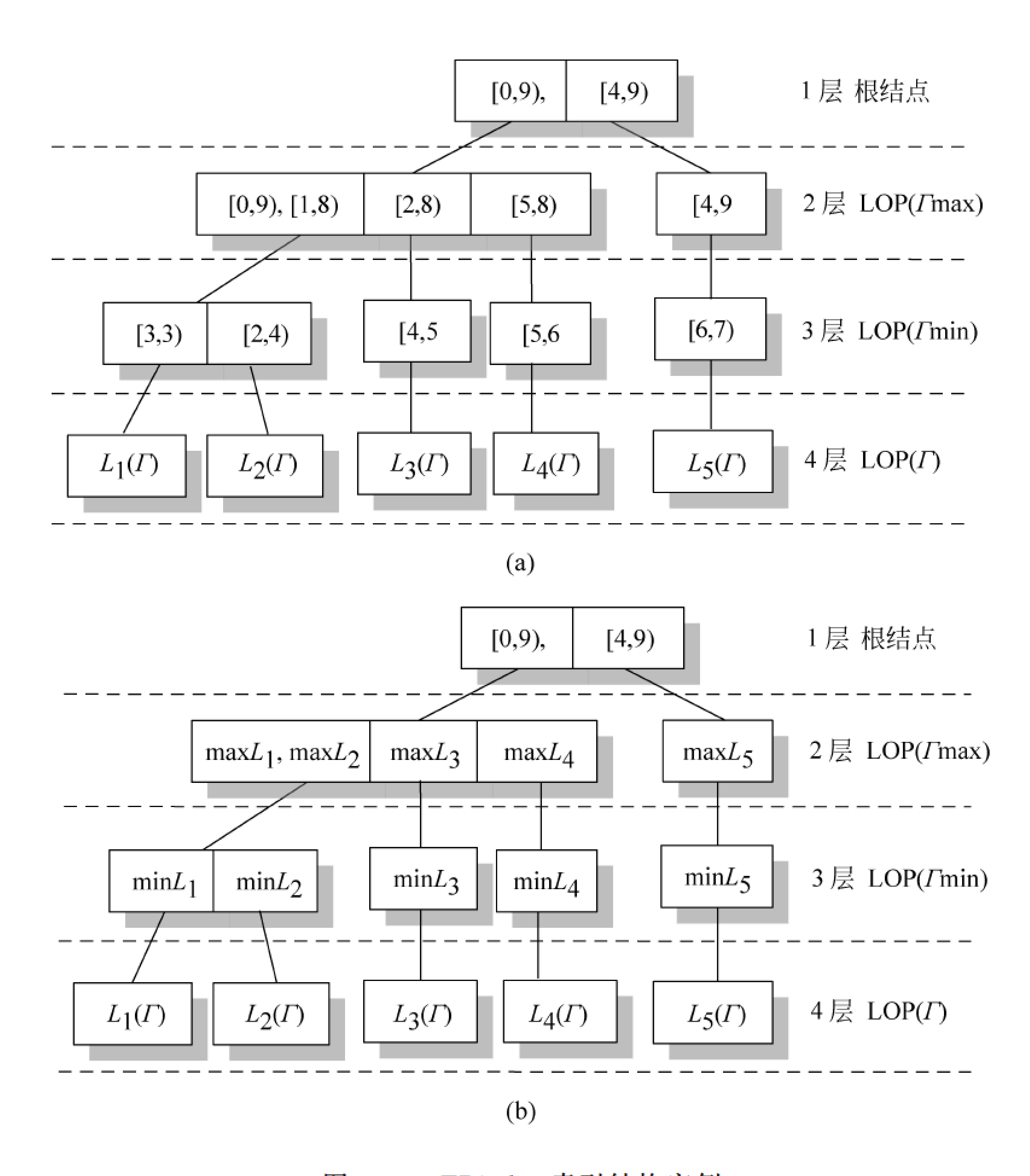


图 10-5　TDindex 索引结构实例

建立索引的目的是有效实现数据的查询，为建立基于 TDindex 的数据查询算法，需要引入下述概念与符号。

【定义 10-6】（线序分枝中点）对于线序分枝 LOB=< $P_1, P_2, \cdots, P_m$ >，max(LOB)= P_1，min(LOB)= P_m，其二分中点记为 mid(LOB)= P_{mid}，其中 mid=(max+min)/2=(1+m)/2，不能整除时取其整数下确界。在不至于混淆时，将不区分标号 mid、max、min 和时间期间 mid，max(LOB)，min(LOB)，$\forall P \in L_i$，P 在 L_i 的位置标号 k 记为 $loc_k(L_i)$。

1. LOB 二分查询

设有查询 Q=[VTs，VTe)，对于包含查询，可以建立下述基于 LOB 的二分查询算法。

【算法 10-2】（二分查询算法）二分查询算法执行步骤如下。

(1) 在[loc_{max}(LOB)，loc_{min}(LOB)]执行二分查找，计算 loc_{mid}(LOB)。loc_{mid}(LOB)是相应二分中点标号。

(2) 若 $Q \subseteq$ mid(LOB)，将< $loc_{max}(L_{i0})$，…，$loc_{mid}(L_{i0})$>放入结果集。

此时，如果当 $loc_{max}(L_{i0}) \neq loc_{min}(L_{i0}) \Rightarrow loc_{max}(L_{i0}) = loc_{mid}(L_{i0})+1$，执行步骤(1)。

否则，执行步骤(3)。

(3) 若 $Q \not\subseteq \text{mid}(L_{i0})$

此时，如果当 $\text{loc}_{\max}(L_{i0}) \neq \text{loc}_{\min}(L_{i0}) \Rightarrow \text{loc}_{\min}(L_{i0}) = \text{loc}_{\text{mid}}(L_{i0}) - 1$，执行步骤(1)。

否则，执行步骤(4)。

(4) 输出结果集。

说明：上述算法也适用于相交查询情形。

基于 LOB 二分查找算法过程如图 10-6 所示。

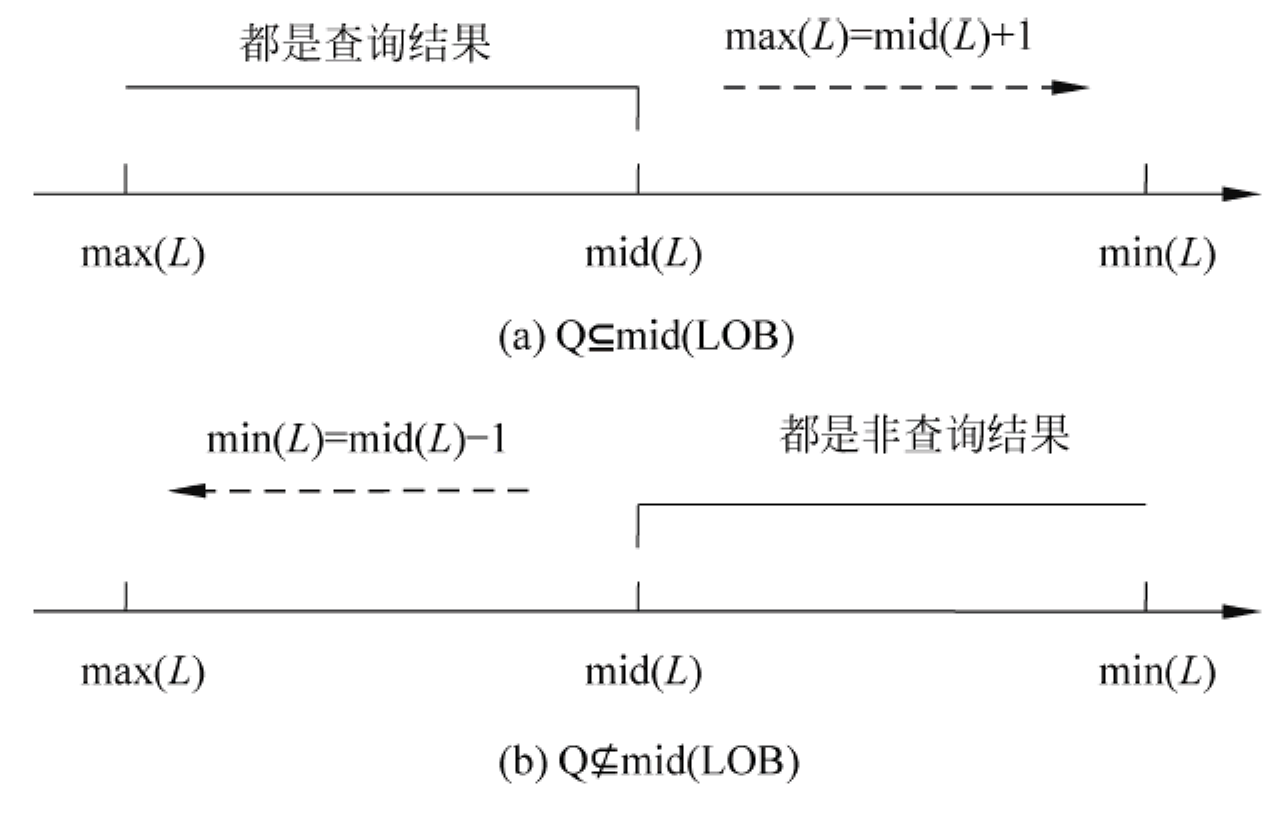


图 10-6　基于 LOB 二分查询过程

2. 数据查询

对于 TDindex 而言，查询过程中所需要的数据都按照线序分枝存储在叶结点中，非叶结点层只具有查询路径的导航作用。

【算法 10-3】 (TDindex 数据查询算法)设 Q 和 Qr 分别为查询要求和查询结果集，Lroot 是根结点中所有元素集合。基于 TDindex 数据查询步骤如下。

(1) 对 $G_k \in \text{Lroot}(1 \leqslant k \leqslant m)$，若 $Q \cap G_k = \varnothing$，G_k 对应 $L_k(\Gamma\text{max})$中所有元素都与 Q 不交。由于 $L_k(\Gamma\text{max})$中元素都是相应 LOB((Γ)的最大元，此时 G_k 对应子树的所有叶结点都不是查询结构。否则，将 G_k 进行标识。取 $\text{Lroot} = \text{Lroot} \backslash \{G_k\}$，继续上述过程直至根结点中最后元素。

对于已标识 G_k，进入 LOP(Γmax)层所对应的子结点 $L_k(\Gamma\text{max})$结点。

(2) 对 $N_i \in L_k(\Gamma\text{max})$，若 $Q \cap N_i = \varnothing$，由于 N_i 是对应 LOB(Γ)中的最大元，此时 N_i 对应子树的叶结点都不是查询结果。否则，对 N_i 进行标识。取 $L_k(\Gamma\text{max}) = L_k(\Gamma\text{max}) \backslash \{S_i\}$，继续上述过程直至 $L_k(\Gamma\text{max})$中无元素。

对于已经标识的 N_i 进入对应的子结点 $L(\Gamma\text{min})$。

(3) 对 $M_j \in L(\Gamma\text{min})$，若 $Q \subseteq M_j$，则 M_j 对应 LOB(Γ)中的所有元素都是查询结果，将此 LOB(Γ)放入 Qr。否则，对 M_j 进行标识。取 $L(\Gamma\text{min}) = L(\Gamma\text{min}) \backslash \{M_j\}$，继续上述过程直至 $L(\Gamma\text{min})$中无元素。

对于已经标识的 N_i 进入 $L(\Gamma\text{min})$对应的子结点 LOB(Γ)。设如此的集合为 LOP_0。

(4) 对 $\text{LOB}(\Gamma) \in \text{LOP}_0$，调用算法 10-2 进行基于 Q 的查找，将所得结果放入 Q_r。如此直至 LOP_0 中无元素。

(5) 输出所得结果集合 Q_r。

由上述算法实际可知，TDindex将时间期间集合上的查询转换到LOP(Γ)之上，进而再转换进线序分枝LOB(Γ)之中。此时对于查询要求Q来说，基于数据查询的基本思想可以表述为：若Q与max LOB(Γ)不交，则LOB(Γ)中所有元素都不是查询结果；如果Q被min LOB(Γ)包含，则LOB(Γ)中所有元素都是查询结果；否则，就实行二分查找。

显然，上述查询算法的时间复杂度主要取决于给定线序划分中线序分枝的个数。由于根据下右优先得到的线序划分具有“最小性”，因此算法10-3应该具有较为理想的查询效率。

3. 查询实例

【例10-3】 对于例10-2所建立的TDindex(Γ)，查询包含$Q_1=[1,2)$的所有时间期间。

(1) 进入根结点，

$Q_1=[1,2)\subseteq \max(L_1(\Gamma\max))=[0,9)$，标识为$G_1$。

因为$Q_1=[1,2)\cap \max(L_2(\Gamma\max)=[4,9)$为空集，所以$L_2(\Gamma\max))$对应子树的叶结点都不是查询结果，需要排除。

进入LOP(Γmax)层中G_1对应的结点$L_1(\Gamma\max)$。

(2) 对于结点$L_1(\Gamma\max)=<[0,9),[1,8),[2,8),[5,8)>$，

因为$[1,8)\cap Q_1=[1,8)\cap[1,2)\neq\varnothing$，连同步骤(1)中[0,9)，标识为$N=\{[0,9),[1,8)\}$，$[2,8)\cap Q_1=[2,8)\cap[1,2)=\varnothing$；所以作为最大元，[2,8)对应子树的叶结点$L_3(\Gamma)$中不存在查询结果。

类似，[5,8)对应子树的叶结点$L_4(\Gamma)$中不存在查询结果。

此时，查询路径只需要沿着[0,9)和[1,8)继续向下行进。

(3) 由N进入到LOP(Γmin)层的结点{[3,3),[2,4)}。

由于[3,3)和[2,4)都不包含$Q_1=[1,2)$，将其标识为$M=\{[3,3),[2,4)\}$。

(4) 对于[3,3)和[2,4)对应的$L_1(\Gamma)$和$L_2(\Gamma)$分别调用算法10-2进行处理。

① 在$L_1(\Gamma)=<[0,9),[0,8),[0,7),[0,6),[1,6),[1,5),[1,4),[1,3),[2,3),[3,3)>$中获得中点mid=[1,6)，$Q_1=[1,2)\subseteq[1,6)$，将$<[0,9),[0,8),[0,7),[0,6),[1,6)>$放入结果集$Q$r。

② 在$<[1,5),[1,4),[1,3),[2,3),[3,3)>$中获得mid=[1,3)，$Q_1=[1,2)\subseteq[1,3)$，将$<[1,5),[1,4),[1,3)>$放入结果集$Q$r。

③ 在$<[2,3),[3,3)>$中获得mid=[2,3)，$Q_1=[1,2)\nsubseteq[2,3)$，排除$<[2,3),[3,3)>$。

因此，得到关于L_1的查询结果是L_1片段：

$$\mathrm{Lp}_1([1,3))=<[0,9),[0,8),[0,7),[0,6),[1,6),[1,5),[1,4),[1,3)>$$

同理，对$L_2(\Gamma)=<[1,8),[1,7),[2,7),[2,4)>$使用二分查找，得到相应查询结果为$L_2$片段：

$$\mathrm{Lp}_2([1,7))=<[1,8),[1,7)>$$

最终得到所需的查询结构为：$\mathrm{Lp}_1([1,3))$和$\mathrm{Lp}_2([1,7))$。

查询过程如图10-7所示，其中灰色框表示实际查询路径，而斜纹框表示包含查询结果的叶结点线序分枝。

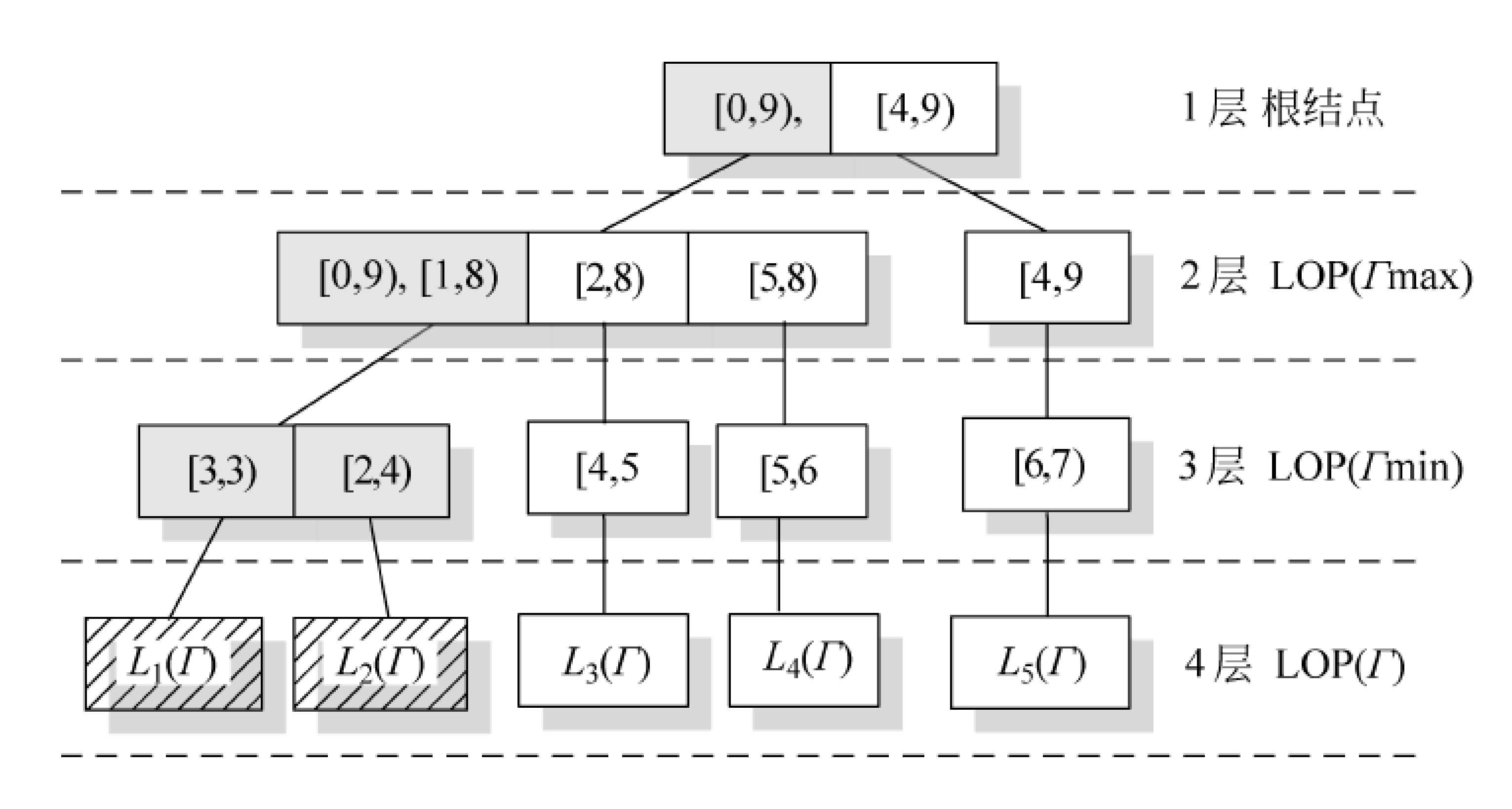


图 10-7 TDindex 数据查询过程

10.1.4 TDindex 增量式更新

更新操作主要为插入操作和删除操作,以下对 TDindex 的增量式插入更新及删除更新进行介绍。

【定义 10-7】 (前驱和后接)设 seg(LOB)为在 VTs-VTe 平面上由 max(LOB)到 min(LOB)遍历 LOB 的轨迹线段。对平面上任意点 v_0,若存在 u_0,满足 $u_0 \in L_0 \wedge \text{VTs}(u_0)=\text{VTs}(v_0) \wedge \text{VTe}(u_0)=\min\{v \mid \text{VTe}(v_0)\leqslant \text{VTe}(u)\}$,则称 u_0 为 v_0 在 L_0 上的直接前驱 $\text{Pre}(v_0)$;若存在 w_0,满足 $w_0 \in L_0 \wedge \text{VTe}(w_0)= \text{VTe}(v_0) \wedge \text{VTe}(w_0)=\min\{w \mid \text{VTs}(v_0)\leqslant \text{VTs}(w)\}$,则称 w_0 为 v_0 在 L_0 上的直接后继 $\text{Suc}(v_0)$。

1. 插入更新

【算法 10-4】 (TDindex 插入更新算法)设 $L_0 \in \text{LOP}$,$L_0 =< v_1,\cdots,v_{i-1},v_i,v_{i+1},\cdots,v_{j-1},v_j,\cdots,v_m >$,待插入元素 u_0。

(1) 若 $\text{LOP}\subseteq \text{ORU}(u_0) \vee \text{LOP}\subseteq \text{OLD}(u_0)$,则构建一个新的 $\text{LOB}=\{u_0\}$。

(2) 若 $u_0 \in \text{seg}(L_0)$,$L_0 = L_0 \cap \{u_0\}$。

(3) 若 $\exists L_0 \in \text{LOP} \wedge (\text{VT}(u_0 \subseteq \min(L_0) \vee \max(L_0) \subseteq \text{VT}(u_0))$, $L_0 = L_0 \cap \{u_0\}$。

(4) 若 $\exists \text{Pre}(u_0) \in L_0 \wedge \text{Pre}(u_0)=v_i \wedge \exists \text{Suc}(u_0)\in L_0 \wedge \text{Suc}(u_0)=v_j$,则构建新 $\text{LOB}=< v_1,\cdots,v_i,u_0,v_j,\cdots,v_m >$,然后将$< v_{i+1},\cdots,v_{j-1}>$作为新的插入元素集合,返回步骤(1),如图 10-8(a)所示。

(5) 若 $\exists \text{Pre}(u_0) \in L_0 \wedge \text{Pre}(u_0)=v_i$,若 $L_0 \cap \text{RU}(u_0)=\varnothing$ 时,构建新 $\text{LOB}=< v_1,\cdots,v_i,u_0,v_i+1,\cdots,v_m >$如图 10-8(b)所示;否则,新增 $\text{LOB}=< v_1,\cdots,v_i,u_0 >$,剩余片段 $L_0 \cap \text{RU}(u_0)=< v_{i+1},\cdots,v_m >$作为新插入元素集合,返回步骤(1),如图 10-8(c)所示。

(6) 若 $\exists \text{Suc}(u_0) \in L_0 \wedge \text{Suc}(u_0)=v_j$,新增 $\text{LOB}=< u_0,v_j,\cdots,v_m >$,片段$< v_1,\cdots,v_{j-1}>$作为新插入元素集合,如图 10-8(d)所示。

(7) 若 $\text{Pre}(u_0)\in L_0 \wedge \text{Suc}(u_0)\in L_0$,若 $\text{RU}(u_0)\cap L_0=\varnothing$,构建新 LOB,如图 10-8(e)所示;否则,构建新 $\text{LOB}=< u_0,v_{i+1},\cdots,v_m >$,$< v_1,\cdots,v_i >$作为新插入元素集合,返回步骤(1),如图 10-8(f)所示。

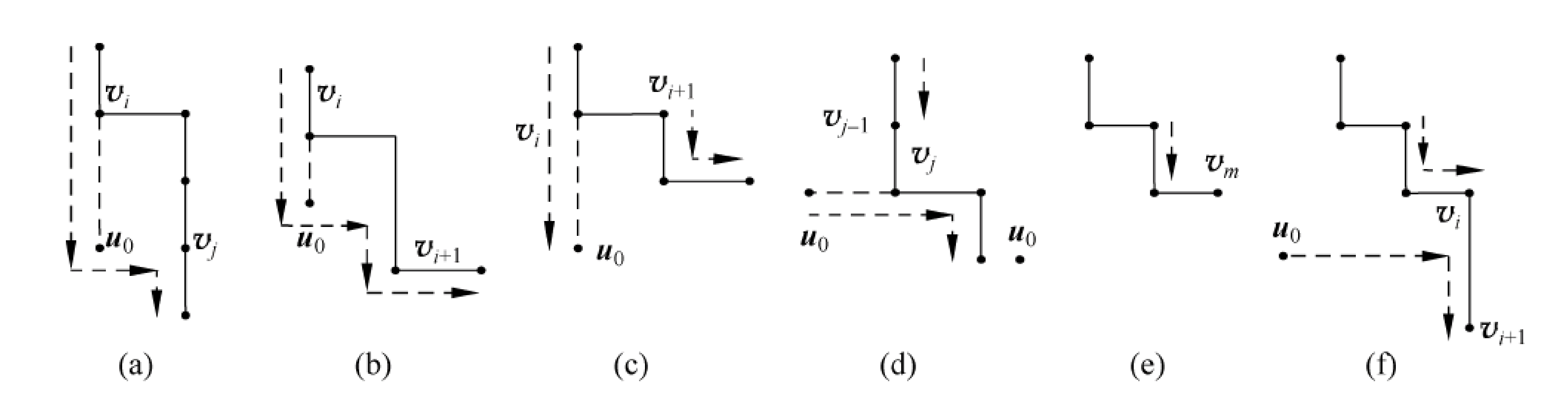


图 10-8　LOB 插入更新

2. 删除更新

【算法 10-5】（TDindex 删除更新算法）设需要删除点为 v_0，u_0、w_0 分别为 v_0 在 LOB 中直接前驱和直接后继。对 LOP 的删除更新，包括以下 3 种情形。

(1) $(VTs(u_0)=VTs(v_0)=VTs(w_0)) \vee (VTe(u_0)=VTe(v_0)=VTe(w_0))$

此时，可直接删除 v_0，然后将 u_0 和 w_0 进行线序分枝拼接，如图 10-9 所示。

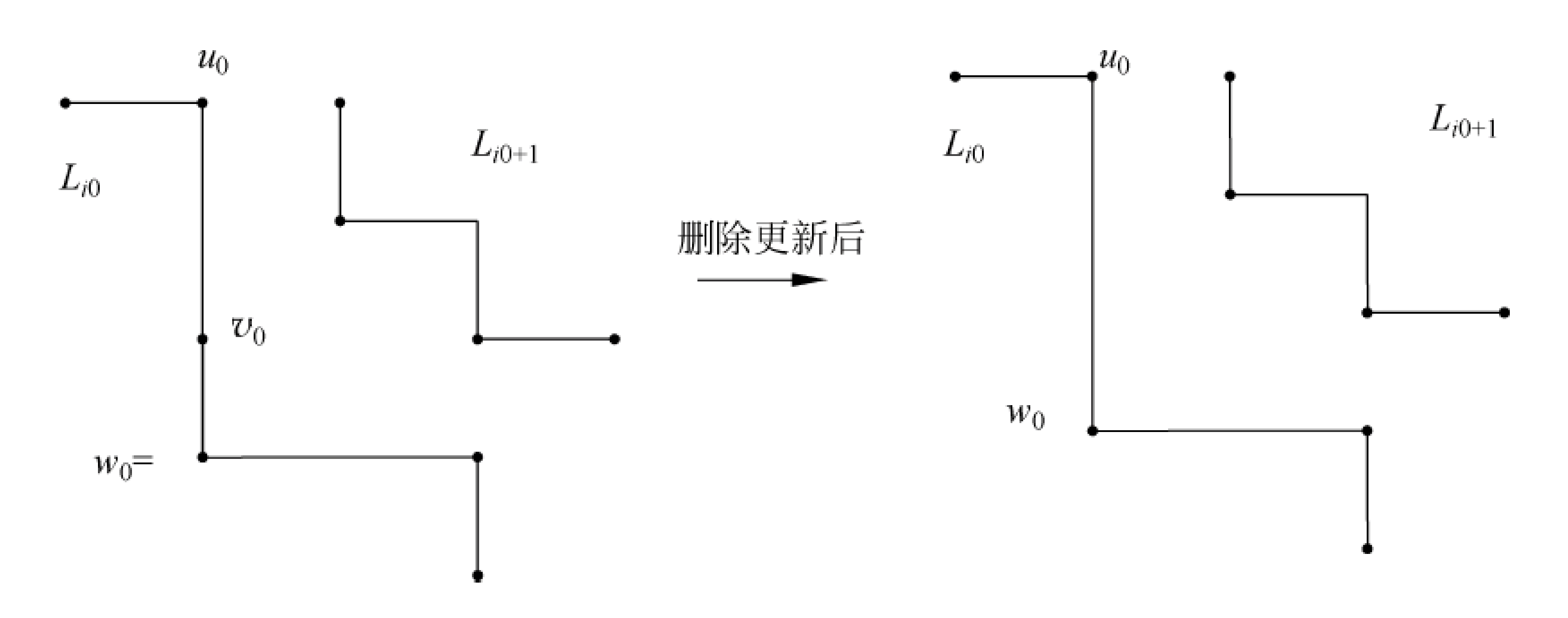


图 10-9　删除更新(1)

(2) $(VTs(v_0)\leqslant VTs(u_0)) \wedge (VTe(w_0)\leqslant VTe(v_0))$，设 $y_0=[VTs(u_0),\ VTe(w_0))$

① 若 $y_0\notin L_{i0+1}$，对 v_0 进行删除，然后通过 y_0 对 u_0 和 w_0 进行线序分枝拼接，L_{i0+1} 不变，如图 10-10 所示。

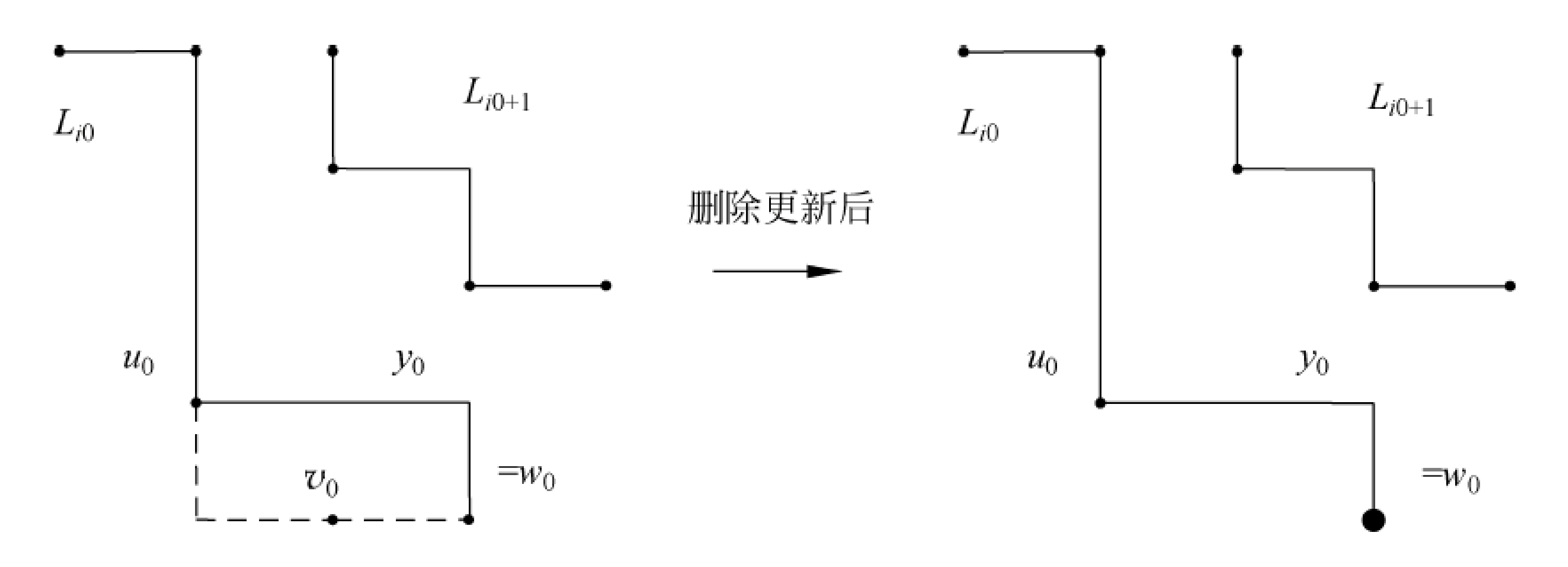


图 10-10　删除更新 Case(2)

② 若 $y_0\in L_{i0+1}$，对 v_0 进行删除，然后通过 y_0 将 u_0 和 w_0 进行线序分枝拼接构建新的 LOB，同时在 L_{i0+1} 中调用 TDindex 删除更新算法对 y_0 进行删除，如图 10-11 所示。

(3) $(VTs(u_0)< VTs(v_0)) \wedge (VTe(w_0)\leqslant VTe(v_0))$，设 $z_0=[VTs(u_0),\ VTe(w_0))$

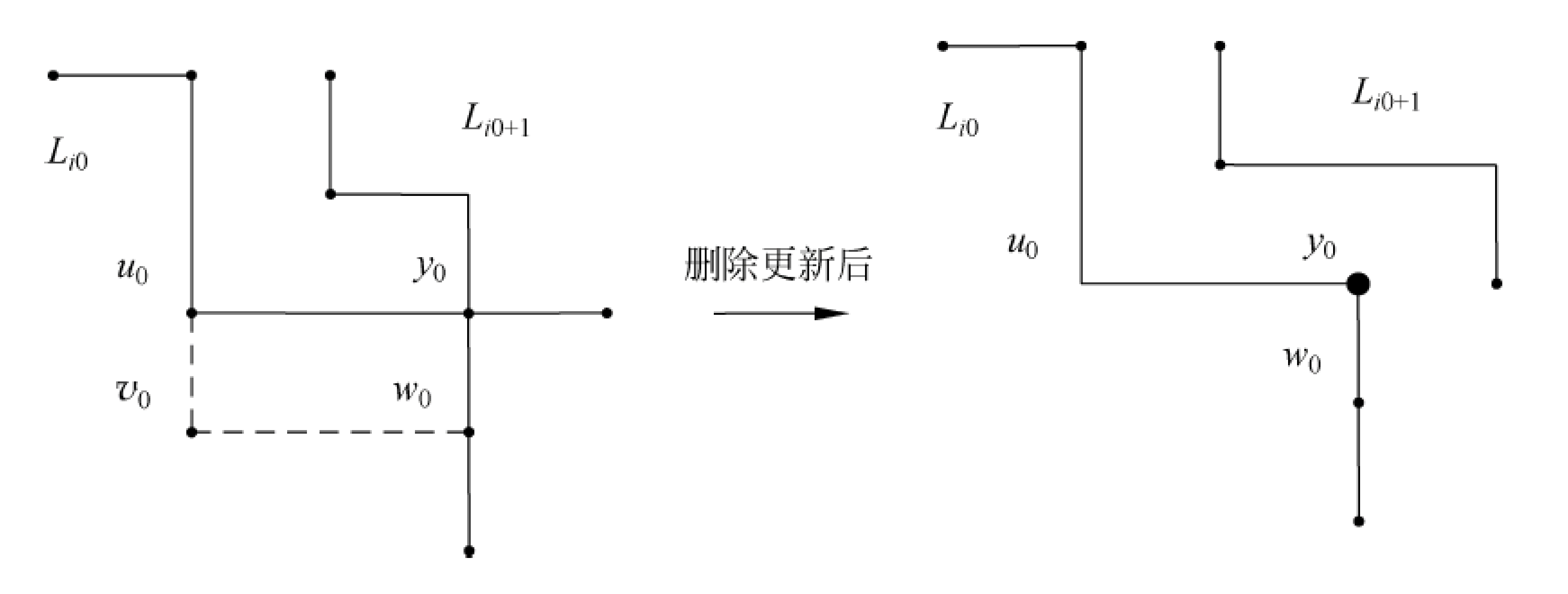


图 10-11　删除更新(3)

① 若 $z_0 \notin L_{i0-1}$，对 v_0 进行删除，然后通过 z_0 对 u_0 和 w_0 进行线序分枝拼接。

如图 10-12 所示，在 L_i。删除 v_0，然后通过 z_0 对 u_0 和 w_0 进行线序分枝拼接构建新的 LOB，而 L_{i0-1} 不变。

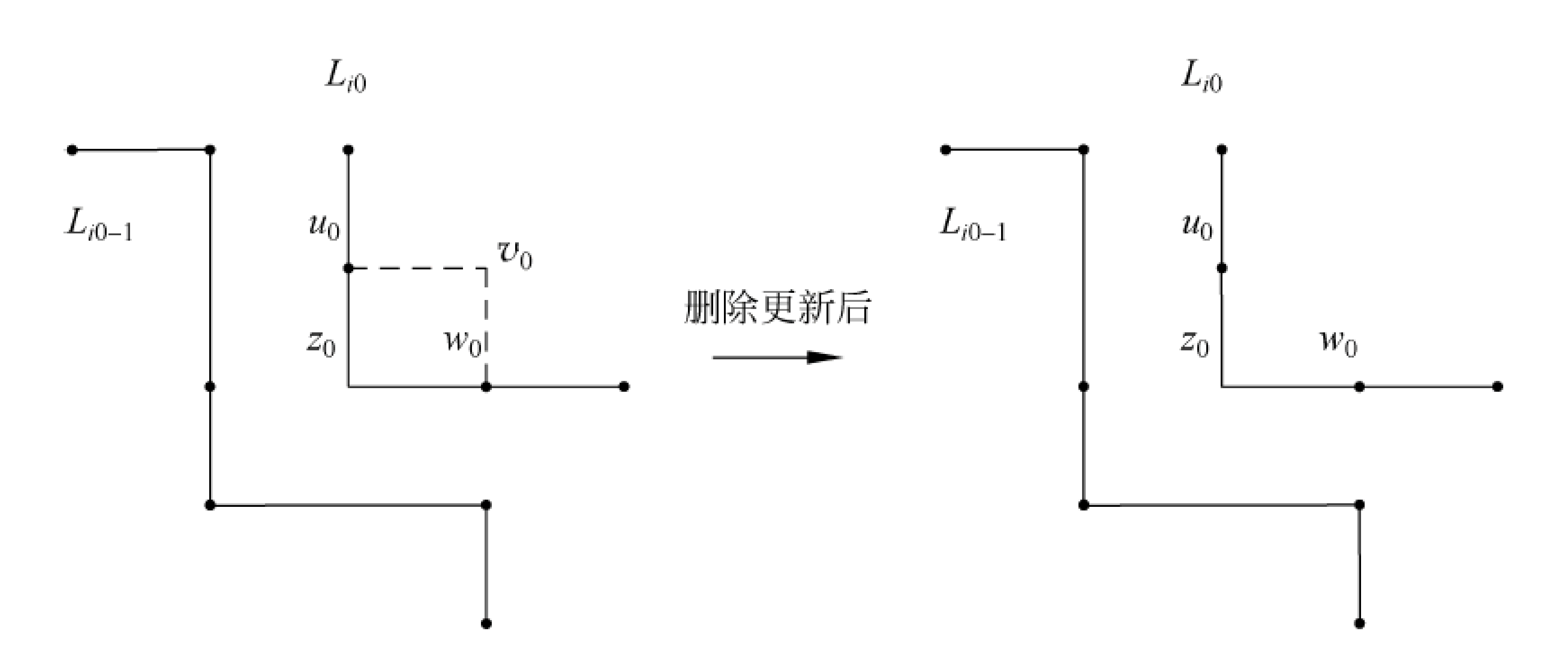


图 10-12　删除更新(4)

② 若 $z_0 \in L_{i0-1}$，对 v_0 进行删除，然后通过 v_0 对 u_0 和 w_0 进行线序分枝拼接构成新的 LOB，L_{i0+1} 不变。

如图 10-13 所示，在 L_{i0} 删除 $v_0=[3,7)$，此时 $z_0 \in L_{i0-1}$，删除 v_0 后，通过 v_0 对 u_0 和 w_0 进行线序分枝拼接构建新的 LOB，L_{i0-1} 不变。

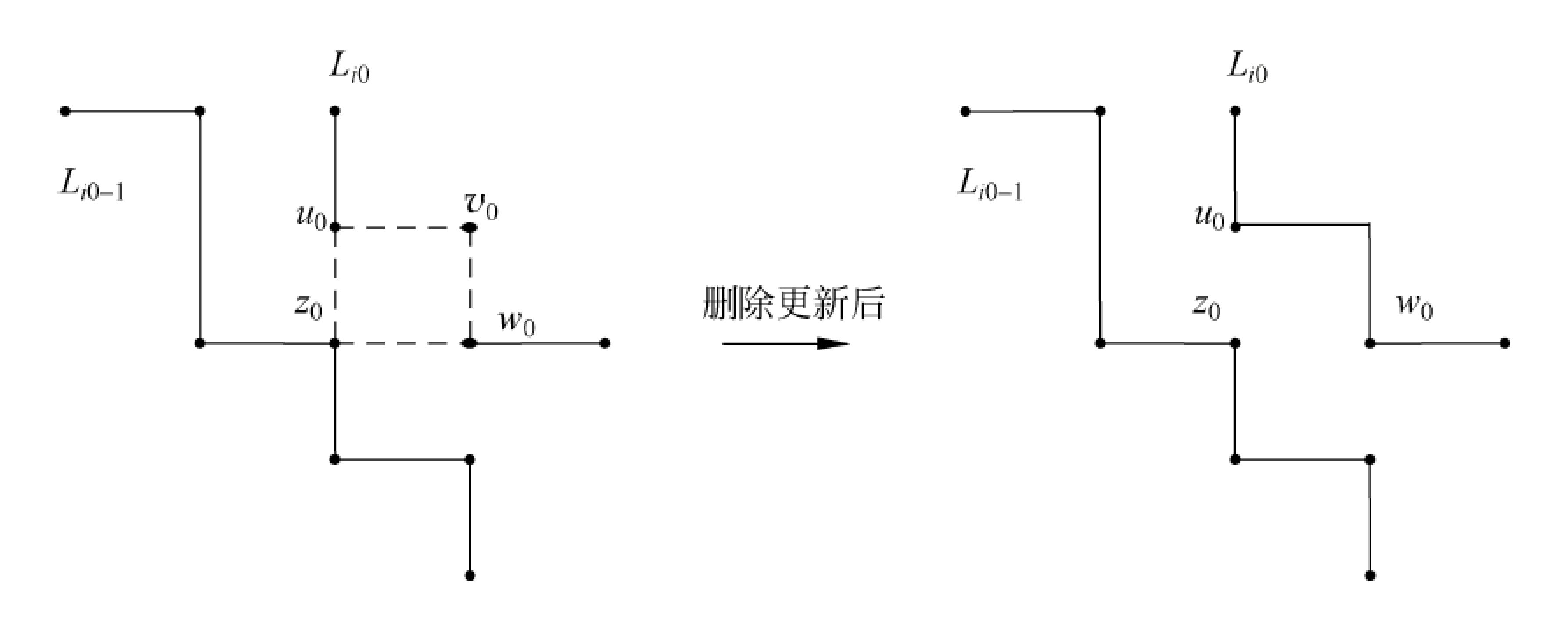


图 10-13　删除更新(5)

10.2 时态 XML 数据索引

作为一种数据结构,时态 XML 数据需要进行以下 3 个方面的描述。

(1) 语义信息:即标签语义,其表现形式为标签路径,不同结构路径可对应相同的标签路径。

(2) 时间信息:即时间标签,通常为时间期间(period)。

(3) 结构信息:父/子关系及衍生的祖先/子孙关系,其表现形式是结构路径;在时态 XML 中,结构与时间信息相互关联,即一个结点的时间信息需要受到其结构信息的制约。例如,父结点的时间期间需要包含子结点的时间期间,右兄弟结点时间期间始点不能小于其左兄弟结点时间始点。

XML 数据结构信息存储和查询比较复杂,是各类相关索引结构讨论的重点。对于时态 XML 数据索引而言,还需要考虑结构信息与时间信息的整合处理。本节讨论一类时态 XML 数据结构 TX-tree,其基本特征是通过 TDindex 索引时间数据,通过广义深度优先编码 GDFc 实现结构信息的存储与查询。

10.2.1 GDFc 编码

通常将 XML 建模为具根分层图,其根结点为虚结点,代表数据操作入口,除根结点外的非叶结点代表元素或属性,叶结点代表元素或属性值。分层图中的边包括结构边和引用边,结构边表明结点之间父/子结构关系,引用边表明结点之间值的引用关系,结点时间标签通常作为相应边标记。

1. 广义深度优先编码 GDFc

【定义 10-8】 (广义深度优先编码)为时态 XML 有根分层图 T_0 中的每个结点 n_0 配置唯一广义深度优先编码:$\text{GDFc}(n_0)=<\text{GDFc}(n_0),\text{gap}(n_0),\text{LevNo}(n_0)>$。

(1) $\text{GDFc}(n_0)$:按照广义深度优先遍历顺序获得的不连续严格单调增编码。

(2) $\text{gap}(n_0)$:与 n_0 更新频率相关的编码间隔。

(3) $\text{LevNo}(n_0)$:n_0 所在层数。

2. GDFc 基本性质

【定理 10-4】 (GDFc 基本性质)定义 10-8 中的 GDFc 编码具有下述基本性质。

(1) $\text{GDFc}(n_0)<\text{GDFc}(n_1)\wedge\text{LevelNo}(n_0)=\text{LevelNo}(n_1)+1$,$n_1$ 是 n_0 的子结点。

(2) $\text{GDFc}(n_0)+\text{gap}(n_0)\geqslant\text{GDFc}(n_1)+\text{gap}(n_1)\wedge\text{LevelNo}(n_0)=\text{LevelNo}(n_1)+1$,$n_1$ 是 n_0 的子结点。

(3) $\text{gap}(n_0)\geqslant\text{gap}(n_1)+\text{gap}(n_2)+\cdots+\text{gap}(n_p)$,$n_0$ 是 $n_1,n_2\cdots,n_p$ 的父结点。

(4) $\text{GDFc}(n_l)+\text{gap}(n_l)\leqslant\text{GDFc}(n_r)\wedge\text{LevelNo}(n_l)=\text{LevelNo}(n_r)$,$n_r$ 为 n_l 右兄弟结点。

证明从略。

【例 10-4】 给定一个时态 XML 实例,其结构与相应 GDFc 编码如图 10-14 所示。

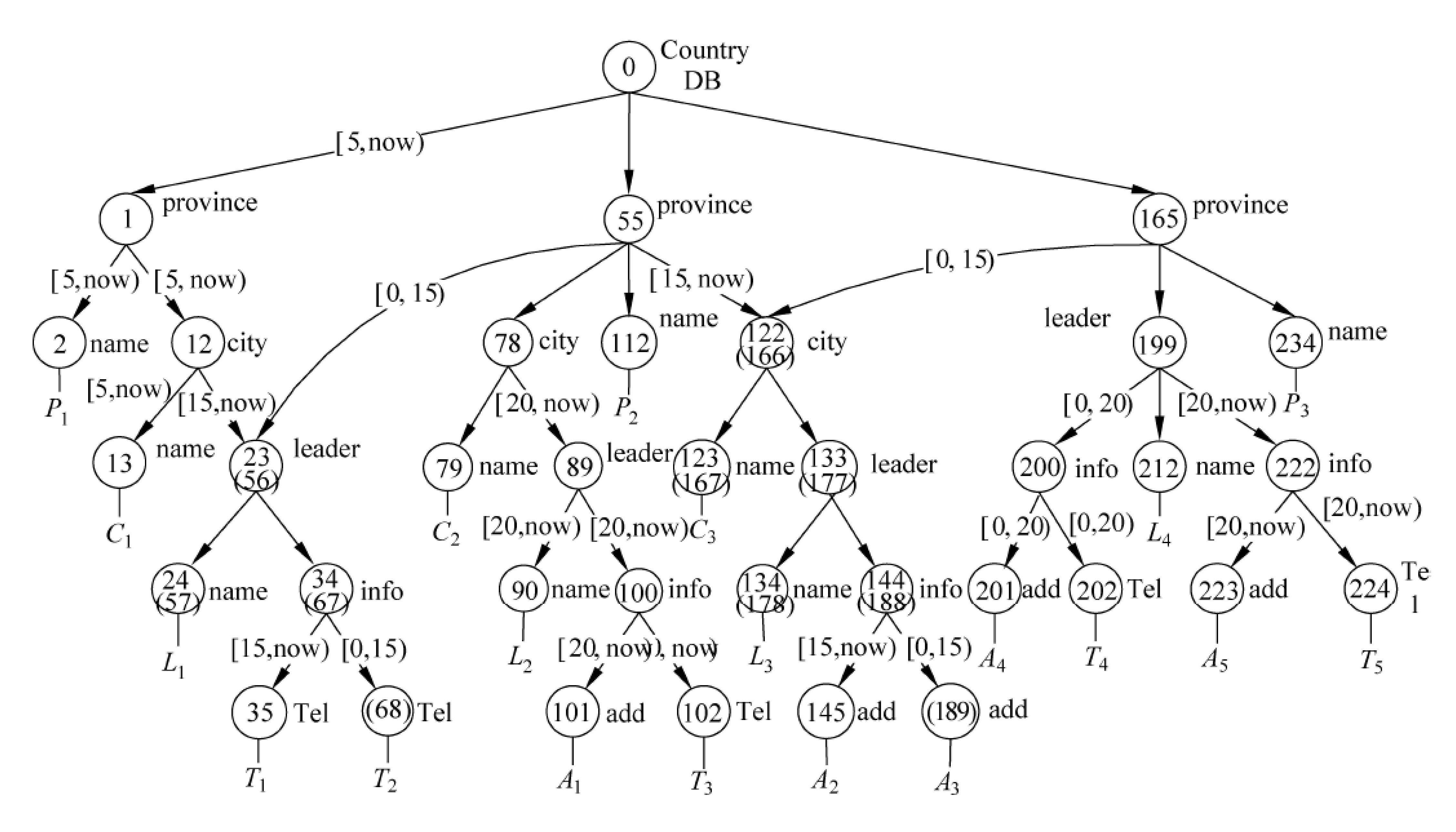


图 10-14　时态 XML 有根分层图和结点 GDFc 编码示例

10.2.2　时态 XML 索引 TX-tree

【定义 10-9】　(时态 XML 索引 TX-tree)TX-tree 表示为如下的三元组。

$$\text{TX-tree}(T_0) = <\text{SNodes}(T_0), \text{TDindex}(T_0), \text{Lnodes}(T_0)>$$

(1) SNodes(T_0)={SNode}：SNode 为 T_0 中具有相同语义标签的数据结点即语义结点。SNodes(T_0)中语义结点依深度优先遍历次序有序。

(2) TDindex(T_0)={TDindex(VT(SNode))}：TDindex(VT(SNode))是对于每个语义结点建立的基于 TDindex 索引树森林，VT(SNode)是 SNode 对应数据的有效时间期间集合。

(3) Lnodes(T_0)：T_0 中按层划分的所有结点 GDFc 编码，即 GDFc 中的 LevNo(n_0)。

【例 10-5】　对例 10-4 中的数据构建 TX-tree(T_0)，相应 SNode、GDFc(SNode)和 TDindex(VT(SNode))如表 10-1 所示，图中时间期间的下标代表处于该时间期间的结点 GDFc，分层 GDFc 编码列表 Lnodes(T_0)如表 10-2 所示。

表 10-1　SNode、GDFc(SNode)和 TDindex(VT(SNode))

Sid	SNode	GDFc(SNode)	TDindex(VT(SNode))
S_0	Country	{0}	{<$[0,now]_0$ >}
S_1	province	{1,55,165}	{<$[0,now]_{55}$,165,[5,now]1 >}
S_2	name	{2,13,24,57,79,90,112,123,134,167,178,212,234}	{<$[0,now]_{79,112}$,$[0,15]_{57,167,178}$>,<$[5,now]_{2,13}$,$[15,now]_{24,123,134}$,$[20,now]_{90,212,234}$>}
S_3	city	{12,78,122,166}	{<$[0,now]_{78}$,$[15,now]_{166}$>,<$[5,now]_{12}$,$[15,now]_{122}$>}
S_4	leader	{23,56,89,133,177,199}	{<$[0,now]_{199}$,$[0,15]_{56,177}$>,<$[15,now]_{23,133}$,$[20,now]_{89}$>}

续表

Sid	SNode	GDFc(SNode)	TDindex(VT(SNode))
S_5	info	{34,67,100,144,188,200,222}	$\{<[0,20]_{200},[0,15]_{67,188}>,<[15,now]_{34,144},[20,now]_{100,222}>\}$
S_6	Tel	{35,68,102,202,224}	$\{<[0,20]_{202},[0,15]_{68}>,<[15,now]_{35},[20,now]_{102,224}>\}$
S_7	add	{101,145,189,201,223}	$\{<[0,20]_{201},[0,15]_{189}>,<[15,now]_{145},[20,now]_{101,223}>\}$

表 10-2　Lnodes(T_0)

level	Lnodes(T_0)
0	{0}
1	{1,55,165}
2	{2,12,78,112,122,166,199,234}
3	{13,23,56,79,89,123,133,177,200,212,222}
4	{24,34,57,67,90,100,134,144,178,188,201,202,223,224}
5	{35,68,101,102,145,189}

10.2.3　TX-tree 数据查询

TX-tree 基于语义划分，在语义结点层面对时态结点进行 TDindex 索引构建，并通过 GDFc 编码实现了“语义”“时间”与“结构”的整合处理。

实际上，从组成要素来看，时态 XML 数据查询实际上需要处理“语义”“时间”与“结构”三方面查询要求。

(1) 语义查询：与结构划分(结构摘要)和时态划分相比，XML 语义结点有数量较少和处理简洁的特性，因此首先处理语义查询可以过滤大量不满足查询条件的结点。TX-tree 中的语义查询通过在语义归并结点中进行遍历查询实现。

(2) 时间查询：需要处理的数据量和复杂程度介于语义查询和结构查询之间，可以作为对语义查询结果的二次过滤。TX-tree 中时间查询通过建立相应的 TDindex 索引结构实现。

(3) 结构查询：时态 XML 数据查询的关键所在，查询操作较为复杂，但通过语义和时间查询，过滤掉了大量不符合查询条件的数据(结点)，查询的数据量较少。TX-tree 中结构查询或结构匹配通过 GDFc 编码实现。

由此可知，TX-tree 在技术层面需要着重处理其中的时间查询部件和结构匹配部件。在时间查询处理方面采用 TDindex 索引框架后，主要课题就是采用 GDFc 编码完成最终的结构查询。因此，需要进一步讨论 GDFc 的相关性质。

1. 结点编码定理

假设 $A_{0i} \in <\text{TDindex}[\text{VT}(A_0)]>$并且满足如下条件：

$$A_{0i} = \max\{A \mid \text{GDFc}(A) \leqslant \text{GDFc}(B_{0k}),$$
$$\forall B_{0k} \in \text{B}[\text{TDindex}(B_0)], \text{LevNo}(A_{0i}) \leqslant \text{LevNo}(B_{0k})\},$$

并设 $C_0 = \min\{c \mid \text{GDFc}(A_{0i}) \leqslant \text{GDFc}(C_0), \text{LevNo}(C_0) = \text{LevNo}(A_{io})\}$即 A_{0i}为编码比 B_{0k}小的最大编码结点，C_0 为 A_{0i}同层右兄弟结点。此时，可以得到下述定理。

【定理 10-5】（结点编码定理）若 A_{0i} 不存在同层右兄弟，则 C_0 为无穷大且成立下述结论：

① 若 $\mathrm{GDFc}(A_{0i})<\mathrm{GDFc}(B_{0k})<\mathrm{GDFc}(C_0)$，则 B_{0k} 为 A_{0i} 子孙结点。

② 若 $\mathrm{GDFc}(A_{0i})<\mathrm{GDFc}(C_0)<\mathrm{GDFc}(B_{0k})$，则 B_{0k} 非 A_{0i} 子孙结点。

证明： ① 使用 $\mathrm{path}(B_{0k})$ 表示路径 $//B_{0k}$，设 B_{0k} 不是 A_0 子孙结点，即 $A_{0i}\notin\mathrm{path}(B_0k)$。因为 $\mathrm{GDFc}(A_{0i})<\mathrm{GDFc}(B_{0k})<\mathrm{GDFc}(C_0)$，$\mathrm{path}(B_{0k})$ 位于 A_{0i} 右侧及 C_0 左侧，与 C_0 定义矛盾。所以①成立。

② 设 B_{0k} 是 A_{0i} 子孙结点，由深度优先遍历可知，C_0 不会位于 A_{0i} 和 B_{0k} 之间，与题设矛盾。证毕。

2. 结构匹配算法

【算法 10-6】（结构匹配算法）设有集合 $<A[\mathrm{VT}(A_0)]>=<\mathrm{TDindex}[\mathrm{VT}(A_0)]>$ 和 $<B[\mathrm{VT}(B_0)]>=<\mathrm{TDindex}[\mathrm{VT}(B_0)]>$，逐一取 $B_{0k}\in<B[\mathrm{VT}(B_0)]>$，在 $<A[\mathrm{VT}(A_0)]>$ 匹配 B_{0k} 的祖先结点，设结果集为 $<A_{0i}//B_{0j}>$。结构匹配算法执行步骤如下。

（1）取 $B_{0k}\in<B[\mathrm{VT}(B_0)]>$，因为 $<A[\mathrm{VT}(A_0)]>$ 单调，在 $<A[\mathrm{VT}(A_0)]>$ 中二分查找定理 10-5 中的 A_{0i}；如果不存在这样的 A_{0i}，执行步骤(4)。

（2）A_{0i} 同层结点集 $\mathrm{Lnodes}(A_{0i})$ 单调增，在 $\mathrm{Lnodes}(A_{0i})$ 上二分查找定理 10-5 中的 C_0；如果这样的点不存在，执行(4)。

（3）由定理 10-5，A_{0i} 与 B_{0k} 存在祖孙结构关系，$<A_{0i}//B_{0j}>=<A_{0i}//B_{0j}>\cap\{A_{0i}//B_{0k}\}$。

（4）$<B[\mathrm{VT}(B_0)]>=<B[\mathrm{VT}(B_0)]>\backslash\{B_{0k}\}$，若 $<B[\mathrm{VT}(B_0)]>\neq\varnothing$，返回步骤(1)，否则，执行步骤(5)。

（5）返回 $<A_{0i}//B_{0j}>$。

3. TX-tree 查询算法

【算法 10-7】（查询算法）假设时态 XML 查询：$Q=A[\mathrm{VT}(A)]//B[\mathrm{VT}(B)]$。TX-tree 数据查询算法执行步骤如下。

（1）对 TX-tree 中的 SNodes 进行语义查询，得到结点列表 $<A_0>$ 和 $<B_0>$，其中 $A=A_0$，$B=B_0$。

（2）通过 TDindex 查询算法分别对 $<A_0>$、$<B_0>$ 进行时态查询，查询结果为 $<A'_0>$ 和 $<B'_0>$，满足 $\mathrm{VT}(A'_0)\subseteq\mathrm{VT}(A)\wedge\mathrm{VT}(B'_0)\subseteq\mathrm{VT}(B)$，且 $<A'_0>$ 和 $<B'_0>$ 按 GDFc 编码排序。

（3）通过结构匹配算法对上述语义和时态查询结构进行结构匹配，以 $<B'_0>$ 为参照，依次将 $<B'_0>$ 中的 B_0 与 $<A'_0>$ 中的每个 A_0 进行结构匹配，输出存在 $A_0//B_0$ 结构关系的结果集 $<A_0//B_0>$ 中。

【例 10-6】 设有查询语句 Q=//city[0,now]//leader[23,now]//Tel，对例 10-4 中的数据进行查询。

① 对“city”进行语义查询，查询结果记为 Label(city)={(12,city),(78,city),(122,city),(166,city)}。

接着，对 Label(city)进行时间查询，结果记为 Lop(city[0,now))={(78,city)}。

② 对“leader”进行先语义后时间的查询，结果分别记为 Label(leader)={(23,leader),(56,leader),(89,leader),(133,leader),(177,leader),(199,leader)} 和 Lop(leader[23,now])={(23,leader),(56,leader),(89,leader),(133,leader),(177,leader),(199,

leader)}。

接着,按照结构匹配算法对 Lop(city[0,now))和 Lop(leader[23,now])进行结构连接,得到(78,city)//(89,leader)。

③ 对"Tel"进行语义查询,结果记为 Label(Tel)={(35,Tel),(68,Tel),(102,Tel),(202,Tel),(224,Tel)}。

接着,按照结构匹配算法对(78,city)//(89,leader)和 Label(Tel)进行结构连接,得到(78,city)//(89,leader)//(102,Tel)。

然后,通过 GDFc=102 映射数据结点 ID,得到 ID=17。

最终查询结果为<Tel ID="17", VT="[20,now]"> T_3 </Tel>。

10.2.4 TX-tree 数据更新

TX-tree 的更新包含 LOP 和 GDFc 更新,其中 LOP 更新通过调用 TDindex 更新算法实现,而 GDFc 更新需要对新增结点进行 GDFc 编码的快速配置。设 CList 为 TX-tree 的 GDFc 集合,A_0 为新插入结点,A_0 父结点为 F_0,通过 GDFC(F_0)可求 F_0 同层后继 B_{0k}。此时,需要在 CList 中确定关于 A_0 满足 GDFc(A_k)<GDFc(A_0)<GDFC(A_{k+1})的前驱 A_k 和后继 A_{k+1}。

【算法 10-8】 (GDFc 更新算法)设 seg_0=(GDFc(F_0), GDFc(B_0)],阈值为 $\alpha(n)$,其中 n=|CList|,NStart 和 NEnd 为 seg_0 的起始和终止位置。GDFc 更新算法执行步骤如下。

(1) Mid=(NStart+NEnd)/2,对 seg_0 执行二分查找直到|seg_0|≤$\alpha(n)$。

① 如果 LevNo(F_0)≤LevNo(Mid),即 $A_{k+1}\notin$(GDFc(NStart),GDFc(Mid)],

seg_0=(GDFc(Mid), GDFc(NEnd)],继续执行步骤(1);否则,转到步骤②。

② 如果 LevelNo(Mid)<LevelNo(F_0),即 $A_{k+1}\notin$(GDFc(Mid),GDFc(NEnd)],seg_0=(GDFc(NStart), GDFc(Mid)]

继续执行步骤(1)。

(2) 遍历查找 seg_0 中首个满足 LevNo(C_0) ≤LevNo(F_0)的结点 C_0,$A_{k+1}=C_0$。

(3) 由 CList 得到 A_{k+1}的直接前驱 A_k,即可得出 GDFc(A_0).

实验表明,TX-tree 的查询性能优于现有的相关索引技术。

10.3 移动对象数据索引

基于时空相点映射的路网移动对象数据索引 pm-tree 是在 MON-Tree 框架内展开讨论的,主要思想是把二维的时空矩形映射为带参数的一维时空相点,然后对相应相点集合建立基于拟序关系的具有时空信息处理能力的 pm-tree 结构,最后结合 2DR*-tree 结构建立起具路网移动对象信息处理能力的数据索引技术。

10.3.1 数据模型与数据结构

路网模型、移动对象时空矩形到时空相点映射和时空相点拟序结构是构建 pm-tree 的基础,需要对这些问题先行讨论。

1. 路网移动对象数据模型

大家已经知道，经典路网移动对象数据模型一般利用二维折线集合表示复杂道路网络结构，其中每条折线表示一条固定的道路（路线），每条道路由一个首尾相接的线段序列组成，线段两端点表示各条道路的交叉点或折线的转折点。下面给出道路与线段的形式化定义。

1）道路和位置点距离参数

【定义 10-10】 （道路或路段，road or line）道路为一个首尾相连的线段序列：

$$R=\{<p_0,p_1>,<p_1,p_2>,\cdots,<p_{n-2},p_{n-1}>,<p_{n-1},p_n>\}$$

其中，$p_i(0\leqslant i\leqslant n)$为二维平面线段的端点，$p_0$ 和 p_n 分别为道路始点和终点，沿 p_0 到 p_n 的方向为 R 的方向。R 上点 p_i 的位置用 p_i 关于 p_0 的沿道路 R（相对）距离参数 $D_i=D(R,\ p_i)$表示。

（2）当 $p_i=p_0$ 时，$D(R,\ p_i)=0$；

（3）当 $p_i\neq p_0$ 时，$D(R,\ p_i)=D(R,\ p_{i-1})+d(p_{i-1},\ p_i)$，$d(p_{i-1},\ p_i)$是 p_{i-1}到 p_i 的欧氏距离（$1\leqslant i\leqslant n$）。

$<p_{i-1},p_i>(1\leqslant i\leqslant n)$称为 R 中由 p_{i-1}到 p_i 的一条路段。

道路 R 实例如图 10-15 所示。

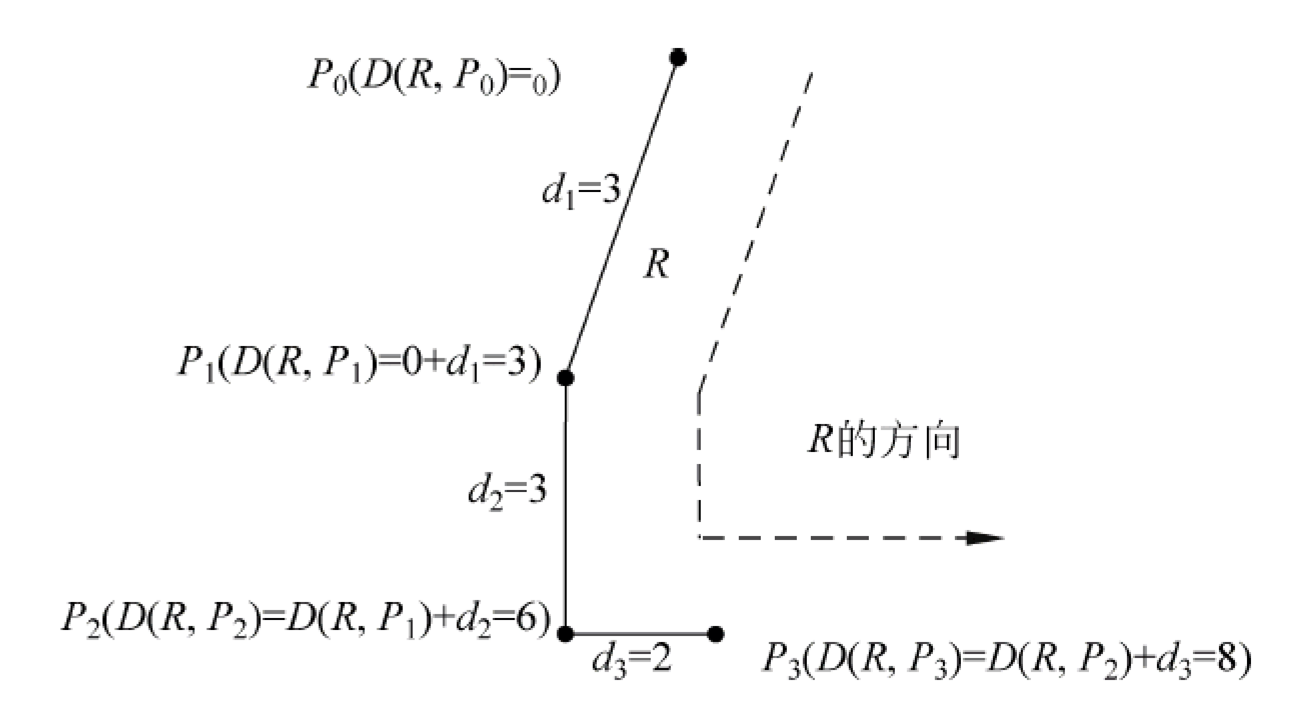


图 10-15　道路 R 和距离参数

以下采用第 8 章中讨论的面向路段的路网模型，也就是说，路网为二元组 RN＝（R_s，N），R_s 是所涉及道路的路段集合，N 是路网中道路交接点及道路始点和终点集合。

2）移动对象时空矩形

【定义 10-11】 （移动对象建模）移动对象 MO 在路网 RN 上运动信息表示为四元组：$M=(R_i,\ D,\ v,\ t)$。

（1）R_i 是 MO 在 NR 中的道路标识。

（2）D 是 MO 在 R_i 上位置。

（3）v 是 MO 速度，当 $v\geqslant 0$ 时表示 m 的移动方向与道路 R_i 方向相同，否则相反。

（4）t 是 MO 位于 D 位置上的时刻。

移动对象轨迹建模：移动对象 MO 在道路 R_i 上运动所产生的运动轨迹可表示为一个四元组序列 $\mathrm{Tr}=<M_0,M_1,\cdots,M_n>$来表示，其相邻两个结点 M_{i-1}和 M_i 组成一个路段（折线段）seg(M_{i-1}，M_i)，其始点和终点空间位置和对应时刻构成的序对分别记为 $M_i-1(\mathrm{star})=(d_{i-1},\ t_{i-1})$和 $M_i(\mathrm{ending})=(d_i,\ t_i)$。

由时间单调递增性，总有 $t_{i-1}\leqslant t_i$ 成立；同时为讨论简便，假设 $d_{i-1}\leqslant d_i$，当 $d_{i-1}>d_i$ 时仅需调换 d_{i-1}和 d_i 的标号即可不会影响相关讨论。

【定义 10-12】 （基于路段移动对象建模）路段上的移动对象可以建模为一个时空矩形。

（1）时空数据矩形（Temporal-Spatial Data Rectangle，TSDR）：移动对象 MO 运行的路段 seg(M_{i-1}，M_i)可表示为一个时空数据矩形 $S_i=(d_{i-1},d_i;\ t_{i-1},t_i)$，其中 S_i 相邻的两条边分别与直角坐标轴 S-axis 和 T-axis 平行，而(d_{i-1}，t_{i-1})和(d_i，t_i)分别为 S_i 左下和右上顶点坐标，如图 10-16 所示。

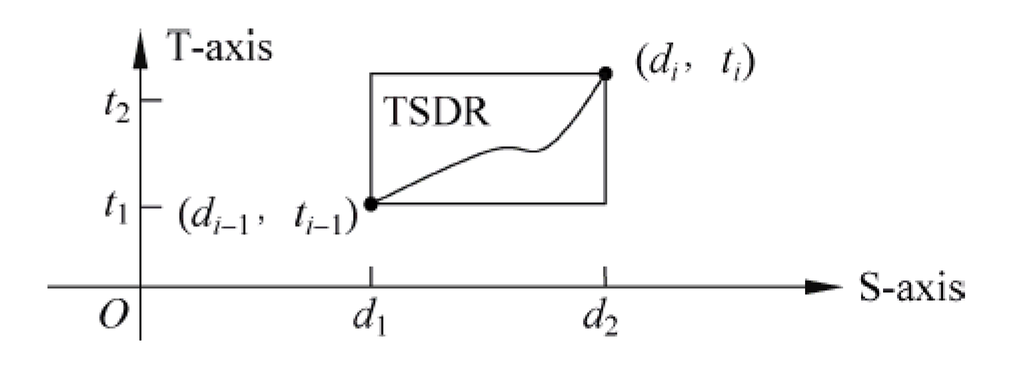


图 10-16　时空矩形 TSDR

为了叙述方便，以下将时空数据矩形简单记为 $S=(d_1,d_2;\ t_1,t_2)$，其中 $d_1\leqslant d_2\wedge t_1\leqslant t_2$。

（2）移动对象数据路段模型：设移动对象 MO 在道路 R 上运动轨迹为 $<M_0,M_1,\cdots,M_n>$，由于每个 M_i 都对应一个时空矩形，因此 M 的轨迹可表示为一个时空数据矩形 TSDR 序列 $<S_1,S_2,\cdots,S_n>$，其中 $S_i=(d_{i-1},d_i;\ t_{i-1},t_i)$，$d_{i-1}$ 和 d_i 分别是 MO 位于点 M_{i-1} 和 M_i 位置时的距离参数，t_{i-1} 和 t_i 分别是 MO 移动到 M_{i-1} 和 M_i 的时刻。(d_{i-1}，d_i)可以看作是 MO 从位置 M_{i-1} 运动到位置 M_i 过程中的空间区间(Space Interval)，(t_{i-1}，t_i)则表示其对应的时间期间(Period)。

【例 10-7】 设有移动对象 MO_1，MO_2，…，MO_{13} 在道路 R 上运动产生的数据及其对应的移动对象数据模型如表 10-3 所示。

表 10-3　移动对象数据建模

Moving Object	Spatial Location	Corresponding Space Interval	Temporal Point	Corresponding Period	Temporal-spatial Data Rectangle
MO_1	0,3,8	(0,3),(3,8)	0,1,5	[0,1), [1,5)	(0, 3; 0, 1)(3, 8; 1,5)
MO_2	0,3,8	(0,3),(3,8)	2,4,7	[2,4), [4,7)	(0,3; 2,4)(3,8: 4,7)
MO_3	0,6	(0,6)	4,8	[4,8)	(0,6; 4,8)
MO_4	3,6	(3,6)	4,8	[4,8)	(3,6; 4,8)
MO_5	3,8	(3,8)	2,7	[2,7)	(3,8; 2,7)
MO_6	3,6,8	(3,6),(6,8)	2,5,6	[2,5), [5,6)	(3,6; 2,5)(6,8; 5,6)
MO_7	0,6,8	(0,6),(6,8)	1,4,6	[1,4), [4,6)	(0,6; 1,4)(6,8; 4,6)
MO_8	6,8	(6,8)	0,3	[0,3)	(6,8; 0,3)
MO_9	0,3	(0,3)	6,8	[6,8)	(0,3; 6,8)
MO_{10}	0,3,6	(0,3),(3,6)	1,2,6	[1,2), [2,6)	(0,3; 1,2)(3,6; 2,6)
MO_{11}	6,3,0	(6,3),(3,0)	1,3,6	[1,3), [3,6)	(3,6; 1,3)(0,3; 3,6)
MO_{12}	0,3,8	(0,3),(3,8)	1,3,7	[1,3), [3,7)	(0,3; 1,3)(3,8: 3,7)
MO_{13}	3,6	(3,6)	5,7	[5,7)	(3,6; 5,7)

2. 时空相点分析

由上述定义可得，移动对象 MO 在道路 R 上的运动轨迹可以用 TSDR 的序列表示，因此仅需对 TSDR 数据进行处理就可得到 MO 的运动轨迹信息。然而 TSDR 作为一个二维

时空矩形，若直接对其进行数据操作，处理效率相对较低。实际上，可以基于TSDR数据的固有特性提出运用数学映射方法把二维的TSDR矩形投影成带参数的一维时空相点，从而提高移动对象运动信息的处理效率。

1）时空相点映射

【定义10-13】 （时空相点映射）时空相点映射（phase points mapping）定义如下。

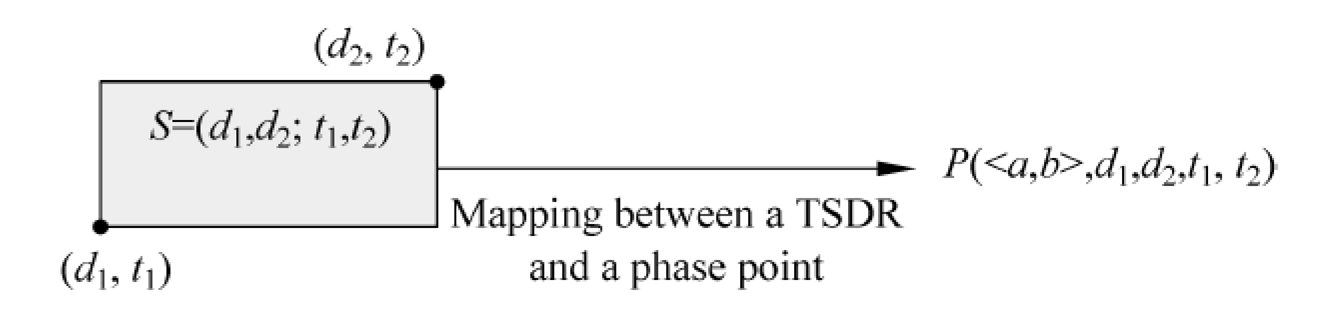
$$S=(d_1,d_2;\ t_1,t_2)\rightarrow P=(<a,b>,d_1,d_2,t_1,t_2)$$

$$a=d_1\times\sqrt{2}+\frac{t_1-d_1}{\sqrt{2}}=\frac{t_1+d_1}{\sqrt{2}}$$

$$b=d_2\times\sqrt{2}+\frac{t_2-d_2}{\sqrt{2}}=\frac{t_2+d_2}{\sqrt{2}}$$

P 为时空数据矩形 S 对应的时空相点（Temporal-Spatial Phase Point，TSPP）；$<a,\ b>$为 P 的时空相点坐标；d_1、t_1、d_2、t_2 为 P 的时空判定参数。

TSDR与相点 P 的映射关系如图10-17所示。



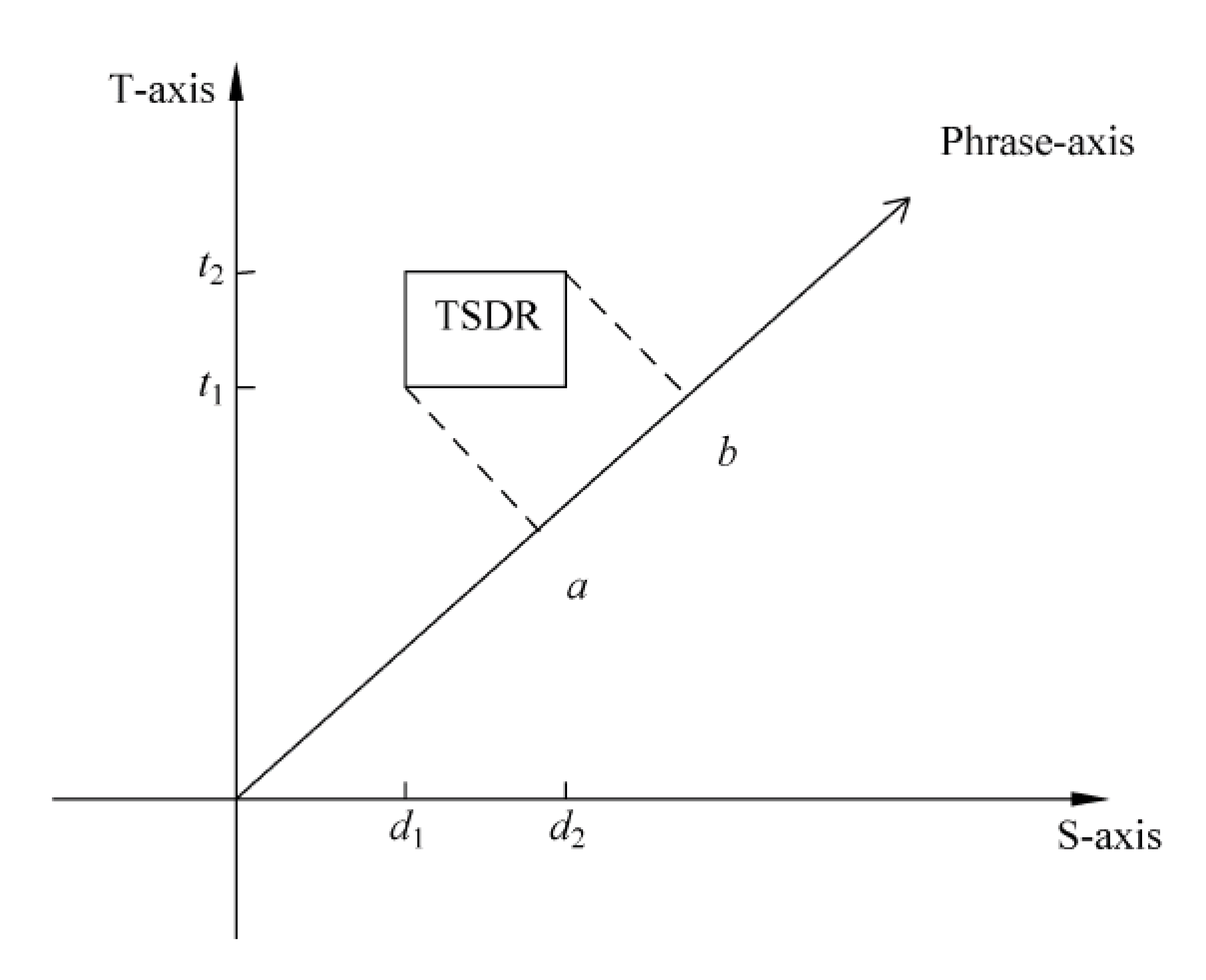


图10-17　TSDR与相点 P 映射关系

对于时空相点 P 的时空相点$<a,b>$可看作相应相平面上的二维点坐标，还可视为S-T平面上相应TSDR在相点轴（phrase-axis）的投影线段(a,b)。由于时空相点坐标$<a,b>$与TSDR的相点轴投影线段(a,b)是一一对应的，以下将不加区别地“混淆”使用$<a,b>$和(a,b)。为了简化计算，不妨把 a、b 分母归一化而都放大$\sqrt{2}$倍，此时即有 $a=t_1+d_1$，$b=t_2+d_2$。

2）面向相点移动对象建模

【定义10-14】 （移动对象时空相点模型，moving object model based on phase point）移动对象在道路R上的运动轨迹可表示为相平面上的一个时空相点TSPP序列 $\sum=<P_1,P_2,\cdots,P_n>$。

【例 10-8】 对于例 10-7 中移动对象 MO 在道路 R 上的运动轨迹数据，其对应时空相点序列 $\sum$ 如表 10-4 所示。

表 10-4　移动对象数据相点表示 $\sum$

Moving Object	Temporal-spatial Data Rectangle	Temporal-Spatial Phase Point
MO_1	(0，3；0，1)(3，8；1,5)	(<0,4>,0,3,0,1),(<4,13>,3,8,1,5)
MO_2	(0,3；2,4)(3,8：4,7)	(<2,7>,0,3,2,4),(<7,15>,3,8,4,7)
MO_3	(0,6；4,8)	(<4,14>,0,6，4,8)
MO_4	(3,6；4,8)	(<7,14>,3,6,4,8)
MO_5	(3,8；2,7)	(<5,15>,3,8，2,7)
MO_6	(3,6；2,5)(6,8；5,6)	(<5,11>,3,6，2,5)，(<11,14>,6,8,5,6)
MO_7	(0,6；1,4)(6,8；4,6)	(<1,10>,0,6,1,4)，(<10,14>,6,8，4,6)
MO_8	(6,8；0,3)	(<6,11>,6,8,0,3)
MO_9	(0,3；6,8)	(<6,11>,0,3,6,8)
MO_{10}	(0,3；1,2)(3,6；2,6)	(<1,5>,0,3,1,2)，(<5,12>,3,6,2,6)
MO_{11}	(3,6；1,3)(0,3；3,6)	(<4,9>,3,6,1,3)，(<3,9>,0,3,3,6)
MO_{12}	(0,3；1,3)(3,8：3,7)	(<1,6>,0,3，1,3)，(<6,15>,3,8,3,7)
MO_{13}	(3,6；5,7)	(<8,13>,3,6,5,7)

$\sum$ 在相平面分布如图 10-18 所示。

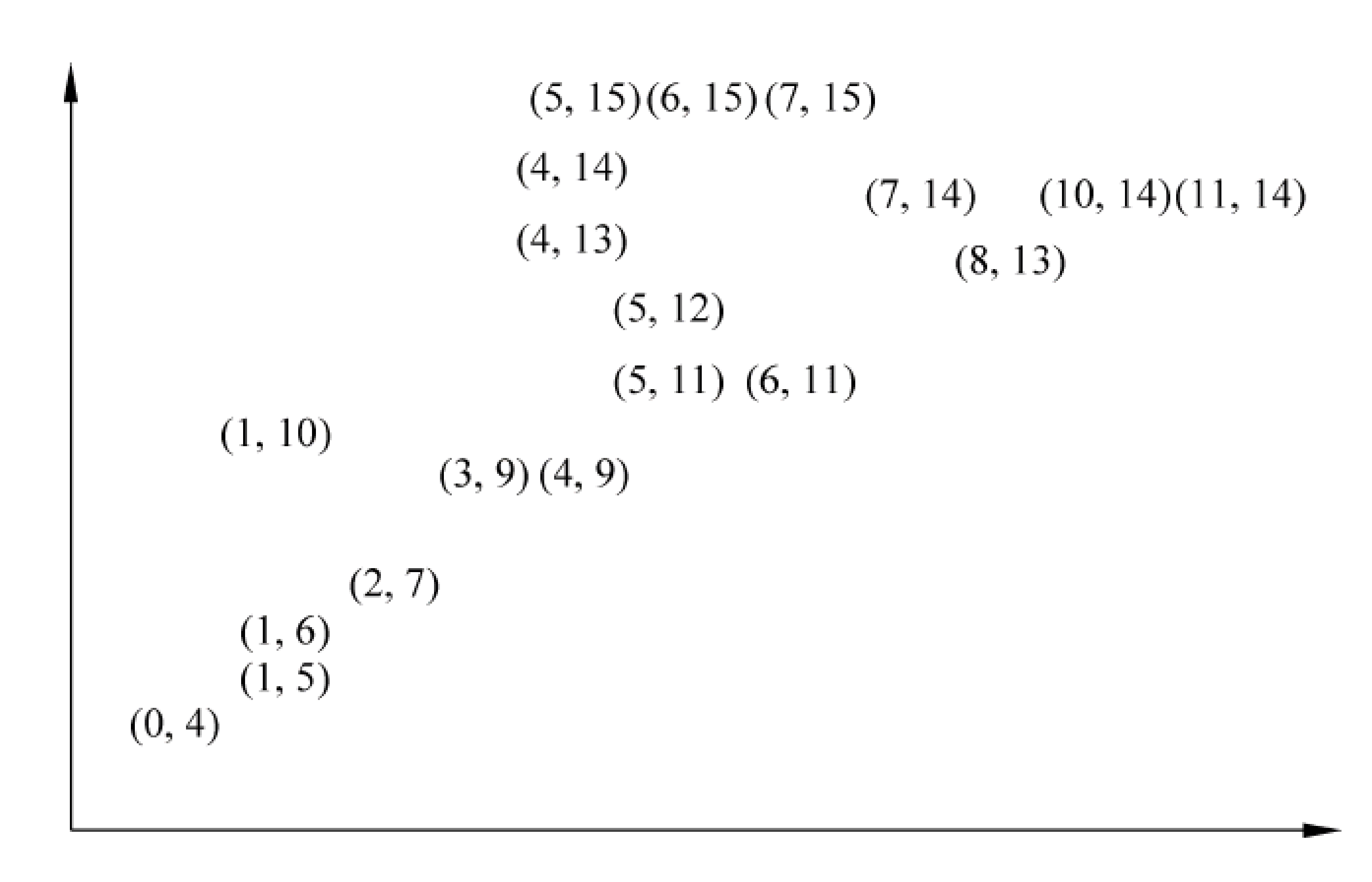


图 10-18　Σ在相平面分布

3）时空矩形与相点区间关系

时空矩形与相点区间具有下述关系定理。

【定理 10-6】 （TSDR 相交关系的相点坐标判定定理）设 $TSDR_i$ 和 $TSDR_j$ 所对应的时空相点分别为 $P_i(<a_i, b_i>, d_{i1}, d_{i2}, t_{i1}, t_{i2})$ 和 $P_j(<a_j, b_j>, d_{j1}, d_{j2}, t_{j1}, t_{j2})$，则成立：

$$TSDR_i \cap TSDR_j \neq \varnothing \Rightarrow (a_i, b_i) \cap (a_j, b_j) \neq \varnothing$$

上述结论的证明由相点映射获得，其正确性也由图 10-19 得到。当 $TSDR_i \cap TSDR_j \neq \varnothing$ 时，$TSDR_i$ 在相点轴上的投影线段 (a_i, b_i) 与 $TSDR_j$ 在相点轴上的投影线段 (a_j, b_j) 必然相交，即 $(a_i, b_i) \cap (a_j, b_j) \neq \varnothing$。

需要注意，上述论述的逆命题是不成立的。

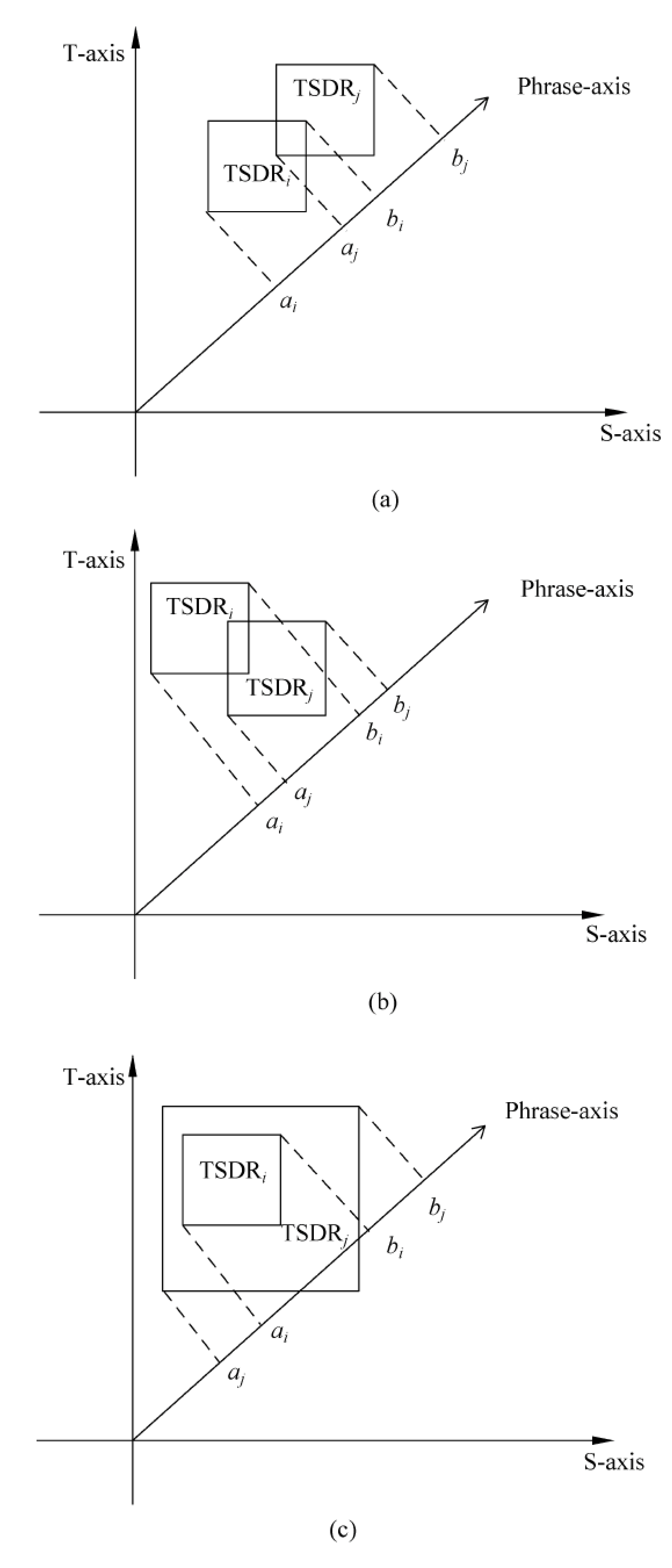
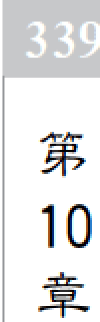


图 10-19 TSDR$_i$ 与 TSDR$_j$ 相交的相点判定

【定理 10-7】 （TSDR 不相交关系的相点坐标判定定理）设 TSDR$_i$ 和 TSDR$_j$ 所对应的时空相点分别为 $P_i(<a_i, b_i>, d_{i1}, d_{i2}, t_{i1}, t_{i2})$和 $P_j(<a_j, b_j>, d_{j1}, d_{j2}, t_{j1}, t_{j2})$，则有：

$$(a_i, b_i) \cap (a_j, b_j) = \varnothing \Rightarrow \mathrm{TSDR}_i \cap \mathrm{TSDR}_j = \cap$$

证明：如图 10-20 所示，当$(a_i,b_i)\cap(a_j,b_j)=\varnothing$时 TSDR$_i$ 和 TSDR$_j$ 的可能分布。此时在相平面中，如果$(a_i,b_i)\cap(a_j,b_j)=\varnothing$，则所有投影到相平面坐标为$(a_i,b_i)$的 TSDR$_i$ 与所有投影到相平面坐标为(a_j,b_j)的 TSDR$_j$ 没有交集，则 TSDR$_i \cap$ TSDR$_j=\varnothing$。



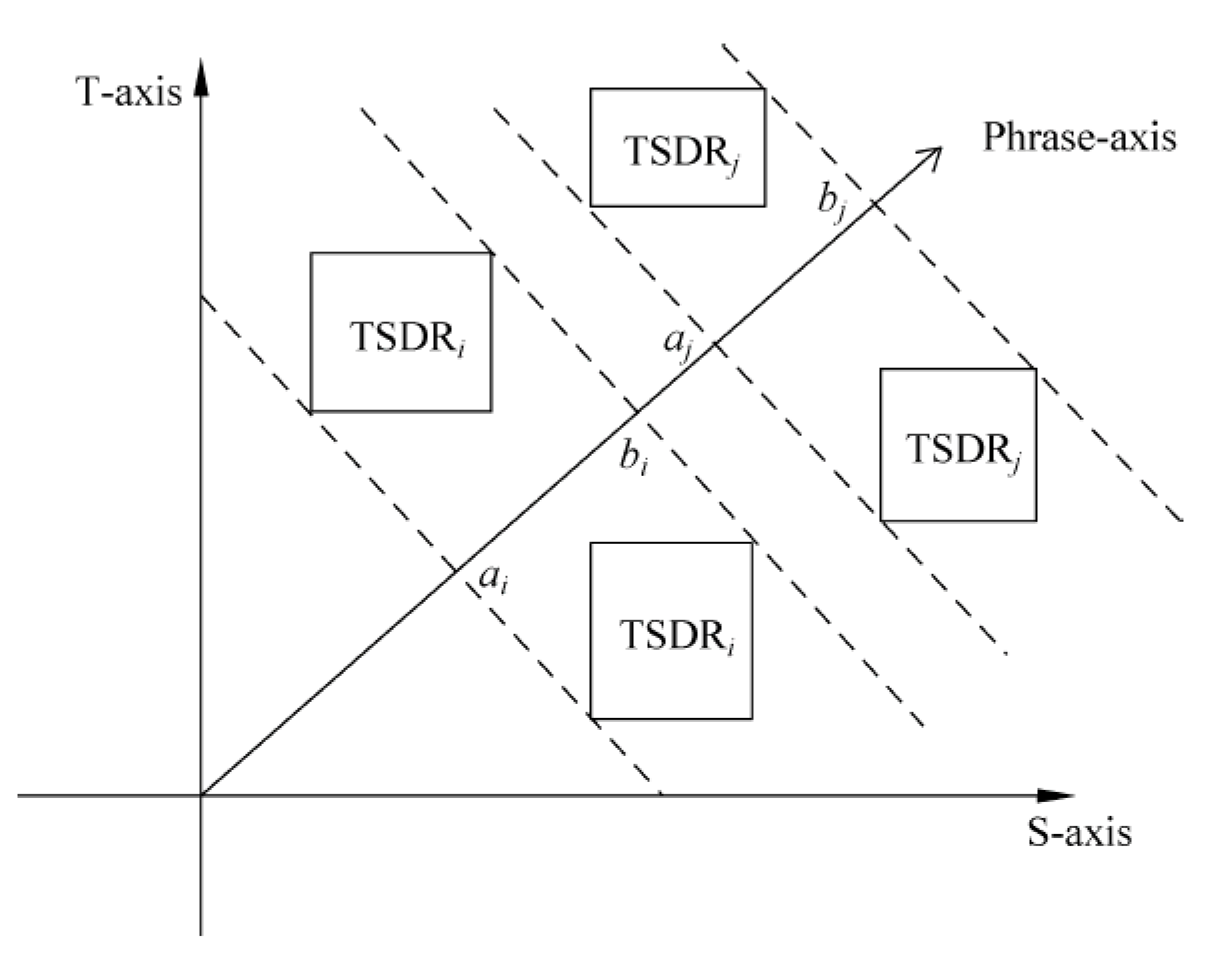


图 10-20　TSDR$_i$ 与 TSDR$_j$ 不相交的相点判定

3. 时空相点数据结构

为叙述方便，在不引起混淆情况下把相点 $P_i(<a_i, b_i>, d_{i1}, d_{i2}, t_{i1}, t_{i2})$与 $P_j(<a_j, b_j>, d_{j1}, d_{j2}, t_{j1}, t_{j2})$分别简记为 $P_i=(a_i,b_i)$和 $P_j=(a_j,b_j)$，并将相点区间(a_i,b_i)和(a_j,b_j)的相交关系$(a_i,b_i)\cap(a_j,b_j)$关系简记为 $P_i\cap P_j$。

如前所述，移动对象 MO 在路网中的运动信息可以表示为相点集合 $\sum=\{P_1,P_2,\cdots,P_m\}$。然后在 $\sum$ 上建立起基于拟序的数据结构，即建立的线序划分 LOP$\left(\sum\right)$。

对于例 10-7 中移动对象数据，对应 LOP$\left(\sum\right)=<L_1,L_2,\cdots,L_m>$：

$$L_1 = <(0,4)>,$$
$$L_2 = <(1,10)(1,6)(1,5)>,$$
$$L_3 = <(2,7)>,$$
$$L_4 = <(3,9)(4,9)>,$$
$$L_5 = <(4,14)(4,13)(5,12)(5,11)(6,11)>,$$
$$L_6 = <(5,15)(6,15)(7,15)(7,14)(8,13)>,$$
$$L_7 = <(10,14)(11,14)>。$$

其中实行 DRFT 算法的过程如图 10-21 所示。

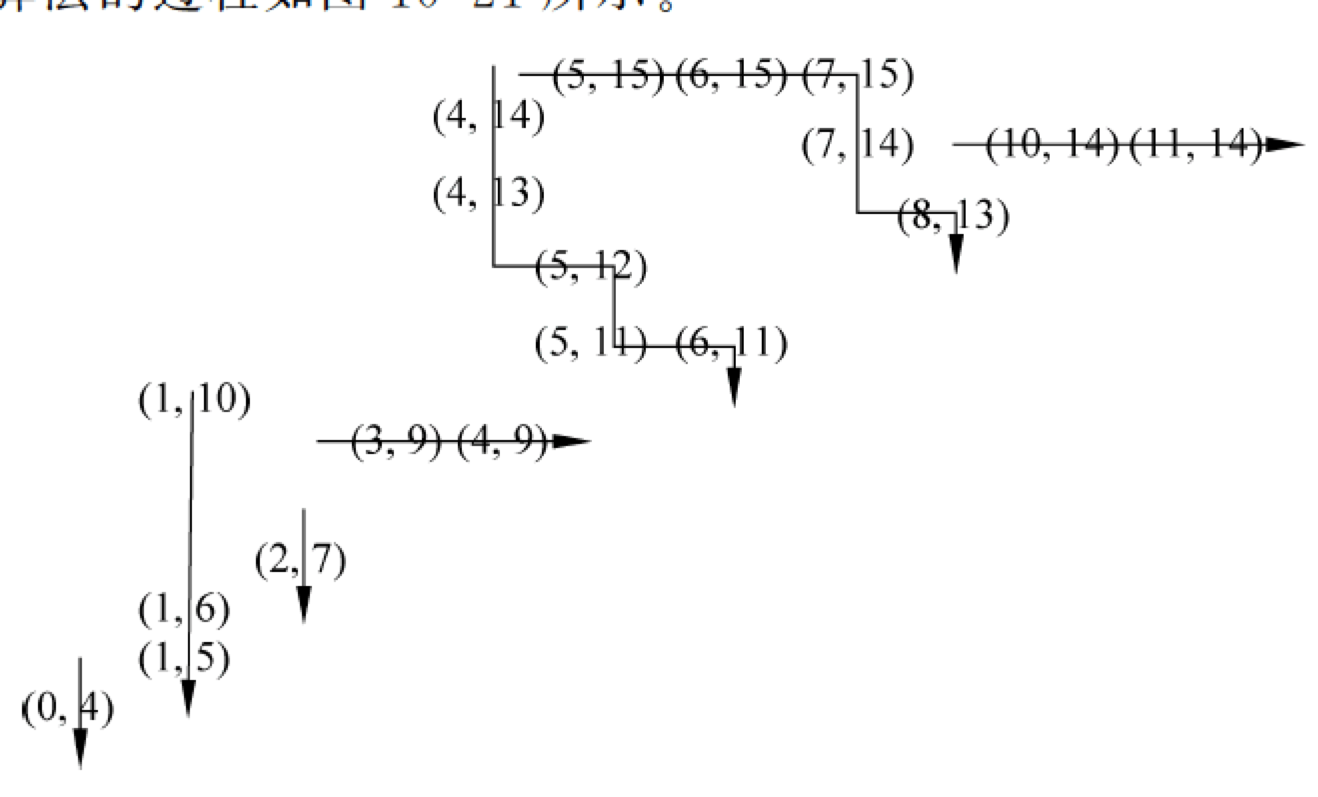


图 10-21　构建 LOP$\left(\sum\right)$

10.3.2 移动对象索引 pm-tree

时空相点移动对象数据索引 pm-tree 的结构采用与 MON-tree 类似的两层索引架构。上层为路网数据处理模块，采用 2DR-tree 存储路网中的道路数据，每个叶结点对应一条道路。下层为道路上移动对象数据处理模块，由一组记录道路 R_i 中移动对象运动信息的 pm-tree 森林组成。

1. pm-tree 基本概念

【定义 10-15】 （移动对象数据索引 pm-tree）pm-tree 表示为如下四元组：

$$\text{pm-tree}=(\text{2DR}^*(\text{Road}),\text{HR};\ \text{pm-tree forest}(\text{MO}),\text{HM})$$

（1）2DR*（Road）：关于路网中道路的 R*-tree 索引。

（2）H_R：连接 2DR*（Road）和 pm-tree forest（MO）的 Hash 映射。

（3）pm-tree forest（MO）：路网中每条道路中的移动对象建立一个 pm-tree，所有道路上的 pm-tree 构成一个 pm-tree 森林。

（4）H_M：连接移动对象与其最新线段的 Hash 映射。

pm-tree 基本架构如图 10-22 所示。

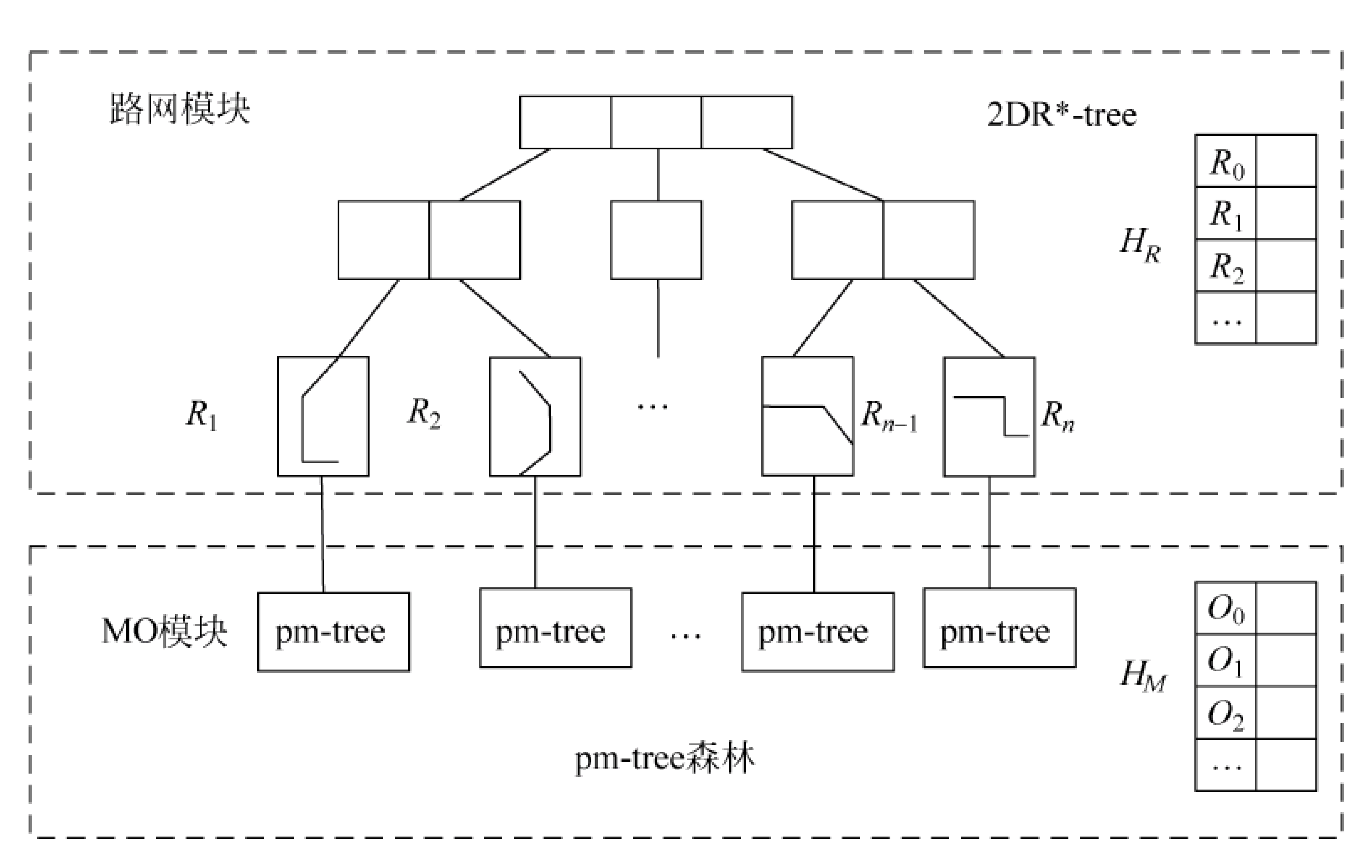


图 10-22 pm-tree 两层框架

2. pm-tree 构建

【定义 10-16】 （pm-tree 构建）pm-tree 是基于 TDindex 的索引结构，可以表示为如下的四元组：

$$\text{pm-tree}=\left(\text{Root},\text{LOP}\left(\sum\max\right),\text{LOP}\left(\sum\right),\text{MO}\right)$$

（1）Root：逻辑层，表示数据操作的入口。

（2）$\text{LOP}\left(\sum\max\right)$：$\text{LOP}\left(\sum\right)$中所有 LOB 中的最大元构成的线序划分。

（3）$\text{LOP}\left(\sum\right)$：$\sum$ 上的线序划分，其中的每个相点均带有一个指向 O-level 对象的指针。

（4）MO：每个相点对应的移动对象构成，存储移动对象具体信息。

pm-tree 的基本架构如图 10-23 所示。

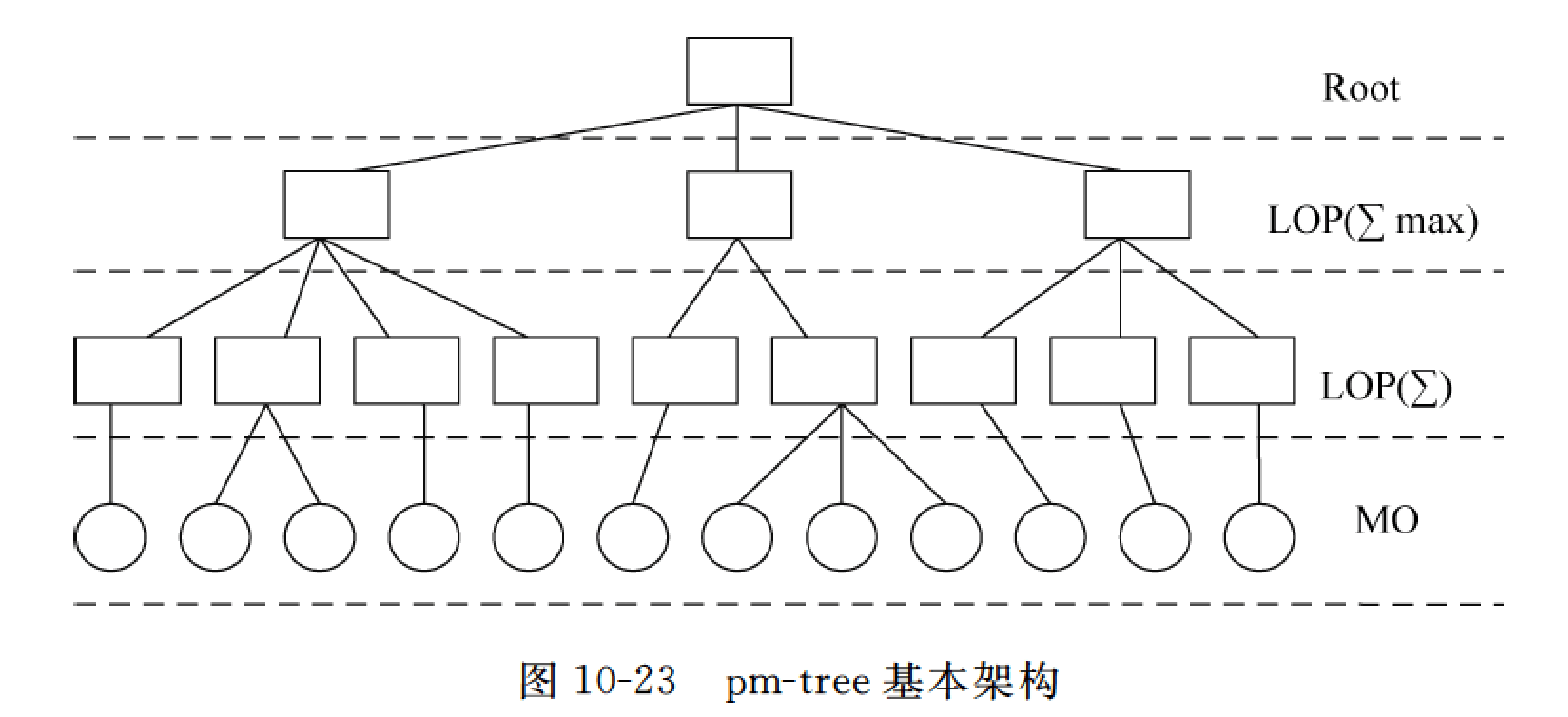


图 10-23　pm-tree 基本架构

【例 10-9】　例 10-7 的移动对象运动数据所构建的 pm-tree 如图 10-24 所示。

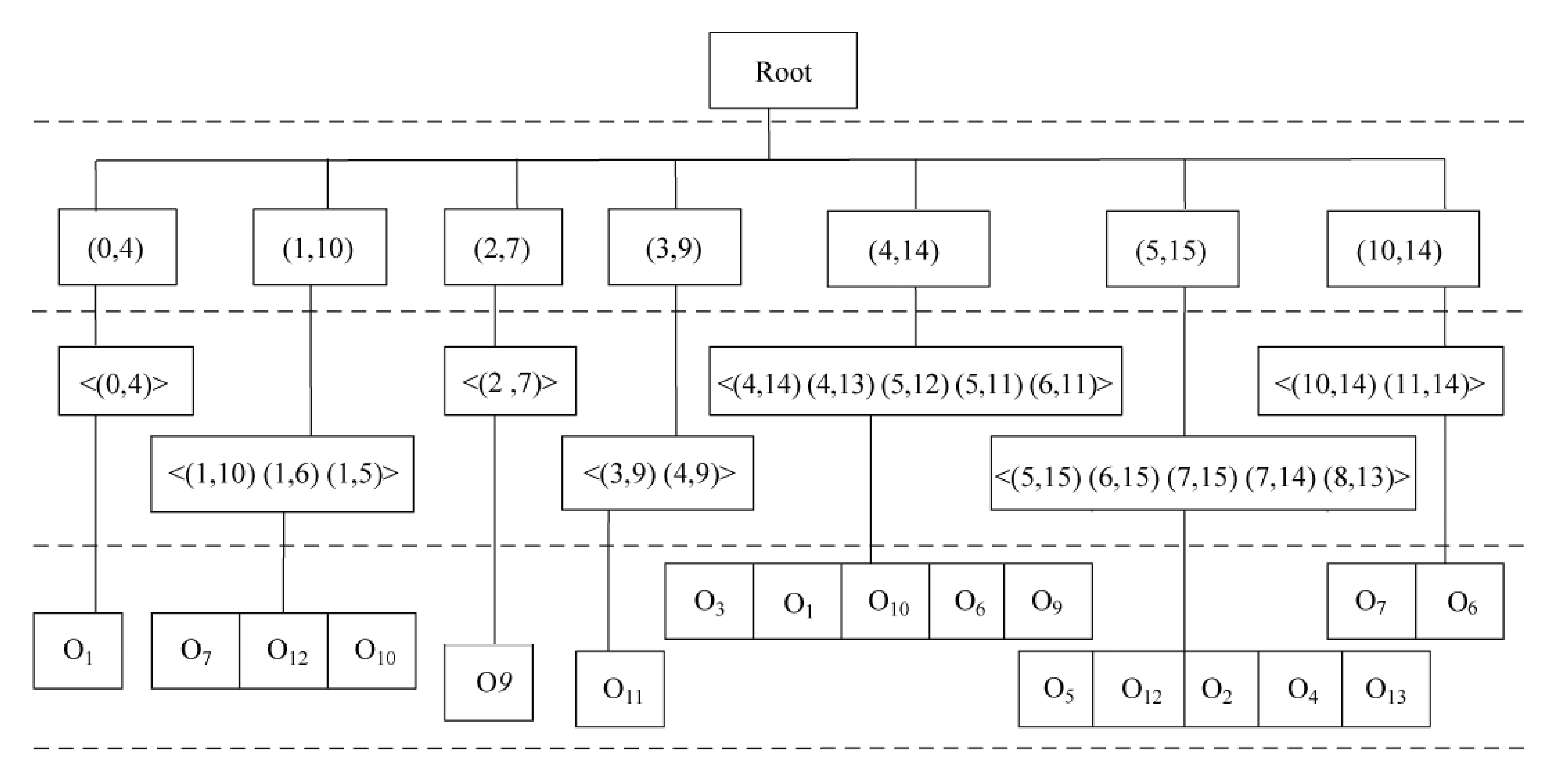


图 10-24　pm-tree 示例

10.3.3　数据操作

下面分别讨论基于 pm-tree 的数据查询和数据更新两种操作。

1. 数据查询

我们已经知道，路网移动对象的查询类型通常一般分为窗口查询、时间片查询和点查询。窗口查询是指给定一个时间间隔和一个空间矩形区域，查找在该时间间隔中位于给定空间矩形区域上的移动对象。时间片查询和点查询均为窗口查询的特殊情况，以下主要讨论 pm-tree 的窗口查询。此时，需要引入下述基本定理。

1）TSDR 相交充要条件

【定理 10-8】　(TSDR 相交关系的相点参数判定充要条件定理)设有如下标记

$$S_i = (d_{i1}, t_{i1}; d_{i2}, t_{i2}) \rightarrow P_i = (<a_i, b_i>, d_{i1}, d_{i2}, t_{i1}, t_{i2}),$$

$$S_j = (d_{j1}, t_{j1}; d_{j2}, t_{j2}) \rightarrow P_j = (<a_j, b_j>, d_{j1}, d_{j2}, t_{j1}, t_{j2}),$$

则 $S_i \cap S_j \varnothing \Leftrightarrow (d_{j1} \leqslant d_{i2} \wedge t_{j1} \leqslant t_{i2}) \wedge (d_{i1} \leqslant d_{j2} \wedge t_{i1} \leqslant t_{j2})$。

证明：

(1) 必要性：S_i 和 S_j 相交可以归结为如图 10-25 所示的 4 种情形，由此可得 $S_i \cap S_j \neq \varnothing \Rightarrow (d_{j1} \leqslant d_{i2} \leqslant t_{j1} \leqslant t_{i2}) \wedge (d_{i1} \leqslant d_{j2} \wedge t_{i1} \leqslant t_{j2})$。

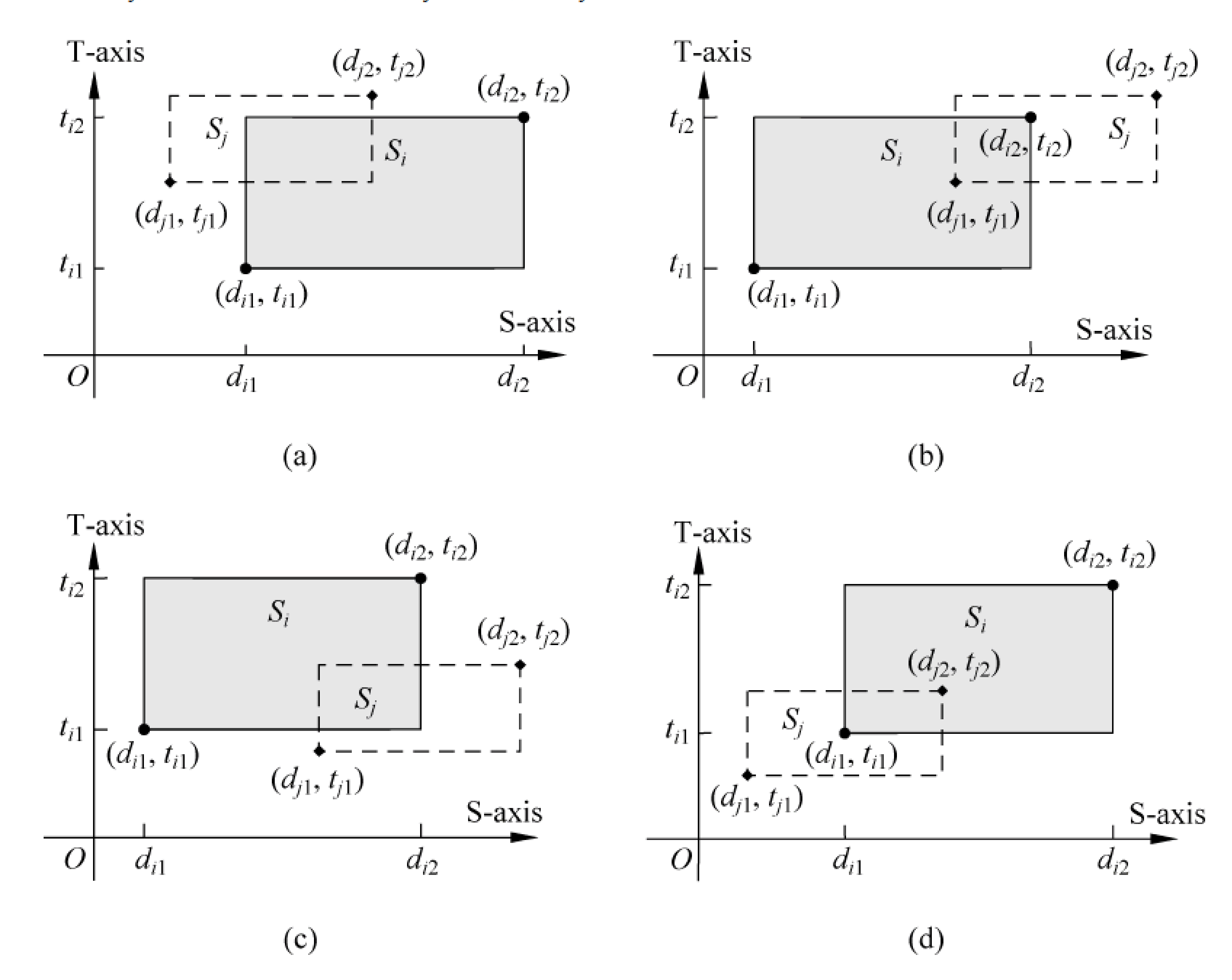


图 10-25　S_i 和 S_j 相交与相点参数关系

(2) 充分性：假设 $S_i \cap S_j = \varnothing$，由于图 10-25 包括了 S_i 与 S_j 相交的各种情形，即当 $S_i \cap S_j = \varnothing$ 时，图 10-25 中各种情形都不出现，此时必有 $(d_{j1} > d_{i2}) \vee (t_{j1} > t_{i2}) \vee (d_{i1} > d_{j1}) \vee (t_{i1} > t_{j1})$，则 $\neg(S_i \cap S_j \varnothing) \Rightarrow \neg(d_{j1} \leqslant d_{i2} \wedge t_{j1} \leqslant t_{i2}) \wedge (d_{i1} \leqslant d_{j1} \wedge t_{i1} \leqslant t_{j1}))$，则原命题 $(d_{j1} \leqslant d_{i2} \wedge t_{j1} \leqslant t_{i2}) \wedge (d_{i1} \leqslant d_{j2} \wedge t_{i1} \leqslant t_{j2}) \Rightarrow S_i \cap S_j \neq \varnothing$ 得证。

当 $(d_{i1} = d_{j2}) \vee (d_{j1} = d_{i2}) \vee (t_{j1} = t_{i2}) \vee (t_{i1} = t_{j2})$ 时，S_i 与 S_j 只有边相交，不满足窗口查询的定义，所以窗口查询的相交判断定理可以简化为：

$$S_i \cap S_j \neq \varnothing \Leftrightarrow (d_{j1} < d_{i2} \wedge t_{j1} < t_{i2}) \wedge (d_{i1} < d_{j2} \wedge t_{i1} < t_{j2})$$

结合上述定理和前述 TSDR 相交关系的相点参数判定定理可得基于时空相点移动对象数据索引 pm-tree 的窗口查询算法。

2) pm-tree 窗口查询

【算法 10-9】 (pm-tree 窗口查询算法)设给定查询窗口 $Q = < x_1, x_2; y_1, y_2; t_1, t_2 >$，查找 $t_1 \sim t_2$ 期间中位于区域 MBR$=(x_1, x_2; y_1, y_2)$ 的移动对象。pm-tree 窗口查询执行步骤如下。

(1) 在 pm-tree 的路网信息处理模块中对 2DR*-tree 从上往下进行搜索，查找与查询窗口 MBR 相交的叶子结点 N_i，并记录 N_i 所对应的道路 R_i。

(2) 对于找到的叶子结点 N_i 通过其指向对象的指针找到道路真实表示，然后把道路与查询窗口 MBR 的相交区域转换成道路中移动对象运动信息处理模块的查询窗口集 $W = \{(d_{11}, t_1; d_{12}, t_1), \cdots, (d_{n1}, t_1; d_{n2}, t_2)\}$。

（3）由映射结构 H_R 查得道路 R_i 所对应的 pm-tree$_i$ 的逻辑入口，在 pm-tree$_i$ 中对于查询窗口集 W 的每个元素（d_{k1}，t_1；d_{k2}，t_1）调用算法 10-10 进行查询并返回查询结果。

3）pm-tree 窗口查询

【算法 10-10】（pm-tree 窗口查询算法）设查询窗口为 $w=<d_1,t_1;d_2,t_2>$，工作空间集为 L，查询候选结果集为 Γ，查询结果集为 sΓ，初始时，$L=\varnothing$，$\Gamma=\varnothing$，$s\Gamma=\varnothing$。pm-tree 窗口查询算法执行步骤如下。

（1）把 $w=<d_1,t_1;d_2,t_2>$ 映射为时空相点 $q=(<a,b>,d_1,d_2,t_1,t_2)$。

（2）进入 pm-tree 的 max-level 层，从左至右进行扫描。

① 若 $q\cap\max(L_i)=\varnothing$，则 L_i 不是查询结果，继续扫描 L_i 的右兄弟结点。

② 否则，$L=L\cup\{L_i\}$，继续扫描 L_i 的右兄弟结点。

（3）进入 pm-tree 的 PPOP-level 层对 L 中的 L_i 进行处理，若 $Q\cap\min(L_i)\neq\varnothing$，把 L_i 记为 $S(L_i)$，转向步骤（4），否则，删除 L_i 的头结点后，对 L_i 执行二分操作，找到 L_i 中最后一个相点 P_k 使得 $P_k\cap q\neq\varnothing$，则 P_k 之后的点与 q 相交均为$\varnothing$，把 L_i 中从 $\max(L_i)$到 P_k 所组成的点集记为 $S(L_i)$；若无这样相点，转向步骤（5）。

（4）$\Gamma=\Gamma\cup\{S(L_i)\}$，$L=L\setminus\{S(L_i)\}$。如果 $L\neq\varnothing$，转向步骤（3），否则，转向步骤（6）。

（5）$\Gamma=\Gamma\cup\max(L_i)$，$L=L\setminus\{S(L_i)\}$。如果 $L\neq\varnothing$，转向步骤（3），否则，转向步骤（6）。

（6）判断 Γ 中的时空相点 $P_j=(<a_j,b_j>,d_{j1},d_{j2},t_{j1},t_{j2})$是否与查询窗口 w 相交，即判断$(d_{j1}<d_2\wedge t_{j1}<t_2)\wedge(d_1<d_{j2}\wedge t_1<t_{j2})$是否成立，若相交则把 P_j 的 O-level 相应移动对象放入 sΓ，算法返回。

4）窗口查询示例

【例 10-10】 设给定 pm-tree 的时空查询窗口 $w=<4,4;5,5>$，例 10-7 中的移动对象轨迹数据的窗口查询过程如下。

（1）把 $w=<4,4;5,5>$映射为时空相点 $q=(<8,10>,4,5;4,5)$。

（2）进入 pm-tree 中 max-level 层，从左至右进行扫描，把满足 $q\cap\max(L_i)\neq\varnothing$ 的 L_2、L_4、L_5、L_6 和 L_7 加入到工作空间集 L 中，此时 $L=\{L_2,L_4,L_5,L_6\}$。

（3）进入 pm-tree 中 PPOP-level 层对 L 中 L_i，依次进行判断：

L_2：因为 $Q\cap\min(L_2)=\varnothing$，所以去掉 $\max(L_2)$对 L_2 剩下结点执行二分查找操作，查找不到相点 $P_k\cap q\neq\varnothing$，因此转步骤（5），把相点（<1,10> 0,6,1,4）加入到 Γ。

L_4：因为 $Q\cap\min(L_4)\neq\varnothing$，所以转向步骤（4），把整条 L_4 加入到 Γ 中。

L_5：因为 $Q\cap\min(L_5)\neq\varnothing$，所以转向步骤（4），把整条 L_5 加入到 Γ 中。

L_6：因为 $Q\cap\min(L_6)\neq\varnothing$，所以转向步骤（4），把整条 L_6 加入到 Γ 中。

（4）此时 Γ={(1,10),(3,9),(4,9),(4,14),(4,13),(5,12),(5,11),(6,11)(5,15),(6,15),(7,15),(7,14),(8,13) }。

Γ 对应的 TSDR 集合为{(0,1；6,4),(0,3；3,6),(3,1；6,3),(0,4；6,8),(3,1；8,5),(3,2；6,6),(3,2；6,5),(6,0；8,3),(0, 6；3,8),(3,2；8,7),(3,3；8,7),(3,4；8,7),(3,4；6,8),(3,5；6,7)}，判断 TSDR 集合中的 $S_i=(d_{i1},t_{i1};d_{i2},t_{i2})$是否满足$(d_{i1}<5\wedge t_{i1}<5)\wedge(4<d_{i2}\wedge 4<t_{i2})$，把满足条件的 S_i 的 O-level 相应移动对象集 sΓ 中。最后 sΓ=$\{O_3,O_1,O_{10},O_6,O_5,O_{12},O_2,O_4\}$。

2. 数据更新

数据更新包括数据删除和插入。由于 pm-tree 着眼于索引移动对象在路网中的历史运动信息，相应更新主要是数据插入，下面仅讨论基于数据插入的索引更新。

1）数据插入两种基本情形

在索引结构中插入结点主要考虑以下两种情形。

（1）插入已有移动对象 MO 的运动信息(x_i, y_i, t_i)。插入过程主要分为以下 5 个步骤。

① 在路网信息处理模块中的 2DR*-tree 查找(x_i, y_i)所在的道路 R_i 并把二维坐标(x_i, y_i)转换为 R_j 距离参数 d_i。

② 借助 H_R 找到 R_j 所对应的 pm-tree 入口并进入道路中移动对象运动信息处理模块。

③ 在 Hash 表 H_M 中查找移动对象 MO 上一个运动信息(d_{i-1}, t_{i-1})，把(d_{i-1}, t_{i-1})和(d_i, t_i)组织成一个新的 TSDR$=(d_{i-1}, t_{i-1}; d_i, t_i)$。

④ 把 Hash 表 H_M 关于对象 MO 的运动信息修改为(d_i, t_i)。

⑤ 索引 TSDR$=(d_{i-1}, t_{i-1}; d_i, t_i)$的插入位置并插入。

（2）插入一个新的移动对象 MO 的运动信息(x, y, t)。插入过程主要分为以下 5 个步骤。

① 在路网信息处理模块中的 2DR*-tree 查找(x, y)所在的道路 R_i 并把二维坐标(x, y)转换为 R_i 距离参数 d。

② 借助 H_R 找到 R_i 所对应的 pm-tree 入口并进入道路中移动对象运动信息处理模块。

③ 在 Hash 表 H_M 中注册对象 MO，并把对象 MO 的运动信息设为(d, t)。

④ 把(d, t)组织成一个新的 TSDR$=(d, t; d, t)$。

⑤ 索引 TSDR$=(d, t; d, t)$的插入位置并插入。

由于两种插入过程相似，下面给出第一种情形时的算法。

2）插入更新算法

【算法 10-11】 (pm-tree 插入更新算法)假设需要将移动对象 MO 产生的运动信息(x, y, t)加入到索引 pm-tree 中，相关算法执行步骤如下。

（1）在 pm-tree 的路网信息处理模块中对 2DR*-tree 从上往下进行搜索，查找包含二维坐标(x, y)的叶子结点 N_i，并记录 N_i 所对应的道路 R_i；通过 N_i 指向对象的指针找到道路真实表示，把二维坐标(x, y)转换为 R_i 距离参数 d。

（2）借助 H_R 找到 R_i 所对应的 pm-tree 入口并进入道路中移动对象运动信息处理模块。

（3）在 Hash 表 H_M 搜索 O，如果存在，读取对象 MO 对应的条目 $entry_k$ 并转向步骤（4），否则，转向步骤（5）。

（4）把 $entry_k$ 中的(d_{i-1}, t_{i-1})和(d, t)组织成一个新的 TSDR$=(d_{i-1}, t_{i-1}; d, t)$，将 Hash 表 H_M 关于对象 MO 的运动信息修改为(d, t)，转向步骤（6）。

（5）在 Hash 表 H_M 中注册对象 MO，并把对象 MO 的运动信息设为(d, t)，把(d, t)组织成一个新的 TSDR$=(d, t; d, t)$，转向步骤（6）。

（6）把 TSDR 映射为时空相点 $P(<a, b>, d_1, d_2, t_1, t_2)$，调用 pm-tree 重构算法把时空

相点 P 插入到 pm-tree 中。

pm-tree 重构算法即是 TDindex 更新算法，增量式重构线序划分 LOP($\sum$) 过程中只需注意调整相应移动对象指针即可。

仿真实验表明，pm-tree 的查询性能优于现有的相关索引技术。

本章小结

就当前而言，只有 RDB 具有成熟的并为人们广泛接受的 DBMS，其他各类新型数据库实现数据查询的有效方式还都需依赖于相应的数据索引技术，也就是说，数据索引和相关数据管理技术基本上都在同步进行。时态数据管理也不例外。

时态数据是显式带有时间标签的数据，时间标签常用和基本形式是时间期间；时态数据索引的关键就在于时间标签集合的索引以及与数据中其他非时态部分的整合。对于所涉及数据对象进行某种意义下的“排序”通常是建立相关数据索引的基本步骤。通过“下右优先”算法，可以对时态数据的时间期间集合建立起时态数据“拟序”结构，即具有自反性和传递性的序关系。时态拟序的作用在于将所涉及时态数据集合进行了基于线序的等价关系划分，从而得以建立起相应的时态数据索引结构 TDindex。

在施行“下右优先”算法过程中，实际上还完成了时间期间始点集合和终点集合在相应线序分枝中的“分组”排序，从而使得基于 TDindex 的数据查询在本质上可以利用“二分查找”的思想。仿真实验表明，此时的查询效率比较理想。能否进行增量式更新是一种索引技术是否有效实用的基本标准之一，TDindex 具有此项功能。

TDindex 作为一种框架思路可以应用于时态关系数据、时态 XML 数据和移动对象数据索引，这里的基本点是对 TDindex 进行具体应用语境中的语义解读。对于时态 XML，相应解读是需要考虑 TDindex 与结构信息的协同整合；对于路网移动对象，相应解读是需要考虑“时空矩形”到“相点区间”的转换。这就是时态 XML 索引 TX-tree 和移动对象索引 pm-tree 讨论的内容。相关实验表明，这两种基于 TDindex 的索引方案是可行的和有效的。

时间期间是一种“区间”结构，因此 TDindex 的思路对于涉及区间数据的索引而言也可能具有一定的参考意义。

主要参考文献

[1] Flavio Rizzolo，Alejandro A. Vaisman. Temporal XML：Modeling，Indexing，and Query Processing [J]. The VLDB Journal，2008.8，17(5)：1179-1212.

[2] Brinkhoff T，Str O. A Framework for Generating Network-based Moving Objects [J]. Geoinformatica，2002，6(2)：153-180.

[3] 叶小平，汤庸，郭欢，等. 时态索引框架技术及其应用[J]. 中国科学：信息科学，2009. 39(12)：1258-1270.

[4] 叶小平，汤庸林，衍崇，等. 时态数据索引 TDindex 研究与应用[J]. 中国科学：信息科学，2015(45)：1025-1045.

[5] 叶小平，陈铠原，汤庸，等，时态 XML 索引技术[J]. 计算机学报，2007(7)：1074-1085.

[6] 叶小平,郭欢,汤庸,等. 基于相点分析的移动数据索引技术[J]. 计算机学报,2011(2):256-274.
[7] 叶小平,汤庸,张智博,等. 语义协同时态 XML 索引研究与实现[J]. 计算机学报,2014(9):1211-1221.
[8] 叶小平,汤庸,林衍崇,等. 时态拟序数据结构研究及应用[J]. 软件学报,2014(25):2587-2601.
[9] 郭欢,叶小平,汤庸,等. 基于时态编码和线序划分的时态 XML 索引[J]. 软件学报,2012,23(8):2042-2057.

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的资源，有需求的读者请扫描下方的"书圈"微信公众号二维码，在图书专区下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010－62770175－4604

资源下载：http://www.tup.com.cn

电子邮件：weijj@tup.tsinghua.edu.cn

QQ：883604（请写明您的单位和姓名）



书圈

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号"书圈"。